Système d'exploitation

Système d'exploitation - Utilisation

Mickael Rouvier

CERI – Avignon Université
mickael.rouvier@univ-avignon.fr

AVIGNON UNIVERSITÉ

Section 1

Gestion des droits des fichiers

Droits d'accès aux fichiers

- **Problématique** : Comment définir les droits d'accès aux fichiers
- **Solution idéal** : Il faudrait définir autant de droits d'accès qu'il y a de :
  - façon d'utiliser un fichier (lecture, écriture, modifications. . .)
  - d'utilisateurs (souvent des centaines)
  - **Impossible** : nombre de combinaisons trop grand
- **Proposition** :
  - 3 façons d'utilisation (appelés droits)
    - droit de lecture (read)
    - droit d'écriture (write)
    - droit d'exécution (ou de traverser un répertoire)
  - 3 classes d'utilisateurs
    - le propriétaire du fichier
    - le groupe auquel appartient le propriétaire
    - tous les autres

Définition utilisateur/groupe

- **Utilisateur** : Toute entité (personne physique ou programme particulier) devant interagir avec un système UNIX est authentifiée sur cet ordinateur par un utilisateur ou user
- **Groupe** : Un utilisateur UNIX appartient à un ou plusieurs groupes. Les groupes servent à rassembler des utilisateurs afin de leur attribuer des droits communs.

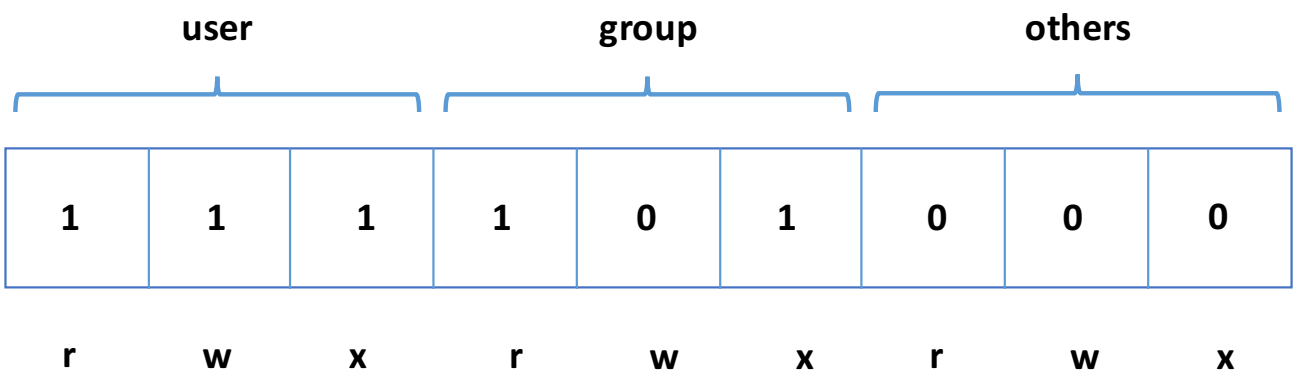
```
invite/> ls -l
-rw-r--r--@ 1 rouvier staff 3613 12 nov 08:02 Archive.zip
-rw-r--r--@ 1 rouvier staff 3613 12 nov 08:02 Archive_cnn.zip
drw-r--r--@ 1 rouvier staff 3613 12 nov 08:02 Chess-Engine
-rw-r--r--@ 1 rouvier staff 3613 12 nov 08:02 test.dic
-rw-r--r--@ 1 rouvier staff 3613 12 nov 08:02 toto.py
```

Affichage des droits

- Affichés sur 10 bits : -rwxrwxrwx
  - 1 : type du fichier
    - - : ordinaire
    - d : répertoire
  - 2 à 10 : droits d'accès/d'utilisation
    - 3 droits : read, write, executable
    - pour les 3 classes d'utilisateurs : user, group, others
- 3 * 3 combinaisons possibles

```
invite/> ls -l
-rw-r--r--@ 1 rouvier staff 3613 12 nov 08:02 Archive.zip
drw-r--r--@ 1 rouvier staff 3613 12 nov 08:02 Chess-Engine
```

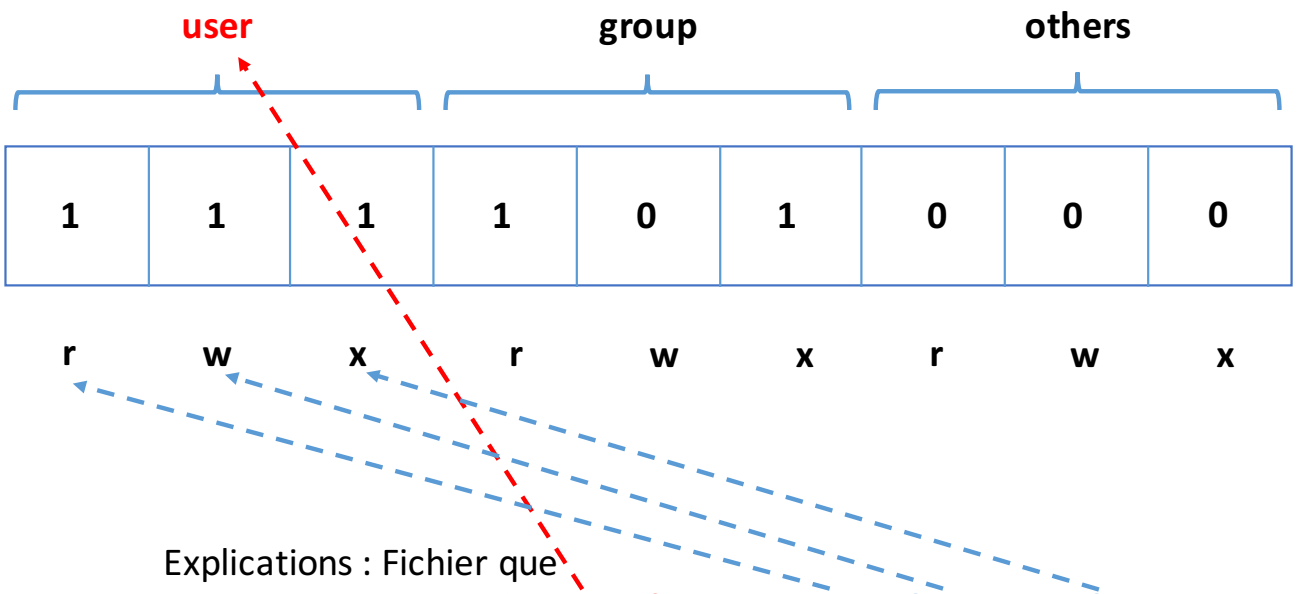
Exemples droits de fichiers



Explications : Fichier que
- le propriétaire peut lire, écrire et exécuter
- ceux du groupe peuvent lire et exécuter
- les autres ne peuvent rien faire dessus

Affichage : -rwxr-x---

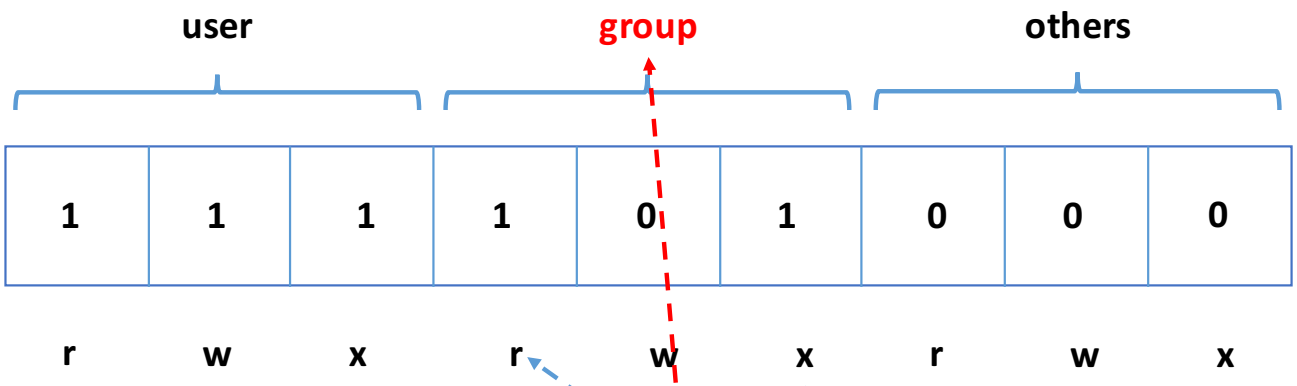
Exemples droits de fichiers



Explications : Fichier que
- le **propriétaire** peut **lire**, **écrire** et **exécuter**
- ceux du groupe peuvent lire et exécuter
- les autres ne peuvent rien faire dessus

Affichage : -rwxr-x---

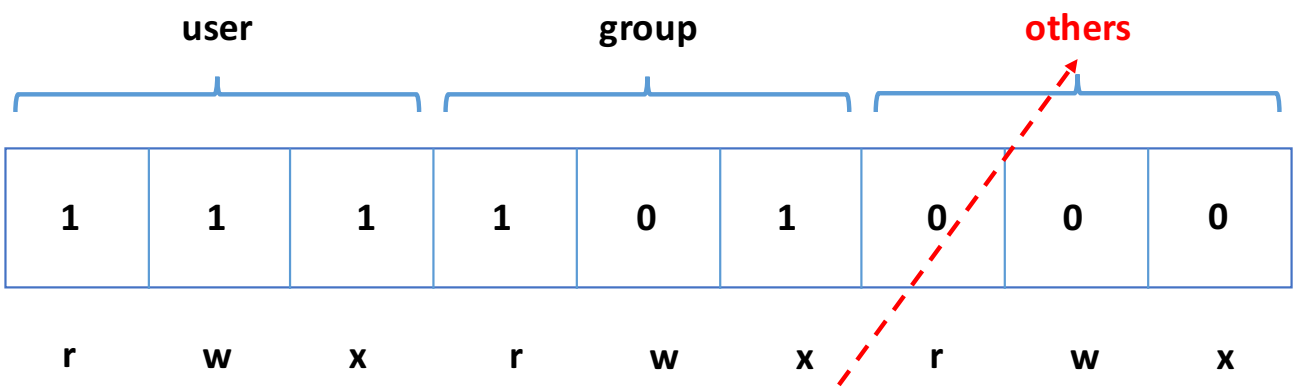
Exemples droits de fichiers



Explications : Fichier que
- le propriétaire peut lire, écrire et exécuter
- ceux du **groupe** peuvent **lire** et **exécuter**
- les autres ne peuvent rien faire dessus

Affichage : -rwxr-x---

Exemples droits de fichiers



Explications : Fichier que
- le propriétaire peut lire, écrire et exécuter
- ceux du groupe peuvent lire et exécuter
- les **autres** ne peuvent rien faire dessus

Affichage : -rwxr-x---

Changement droits d'un fichier

- Commande **chmod** (**ch**ange **mod**e)
- Il existe deux manières d'utiliser la commande :
  - **Utilisation des symboles** : chmod [ugoa] [+-=] [rwx] filename
  - **Utilisation du nombre octal** : chmod OctalMode

Changement droits d'un fichier - Utilisation des symboles

- **chmod [ugoa] [+-=] [rwx] filename**
- Classe :
  - **a** appliqué à tous (défaut)
  - **u** appliqué au propriétaire (user)
  - **g** appliqué au groupe
  - **o** appliqué aux autres (others)
- Opératios
  - **+** ajout de droits
  - **+** retrait de droits
  - **=** affectation de droits
- Droits
  - **r** droit de lecture (read)
  - **w** droit d'écriture (write)
  - **x** droit d'exécution (ou de traverser un répertoire)

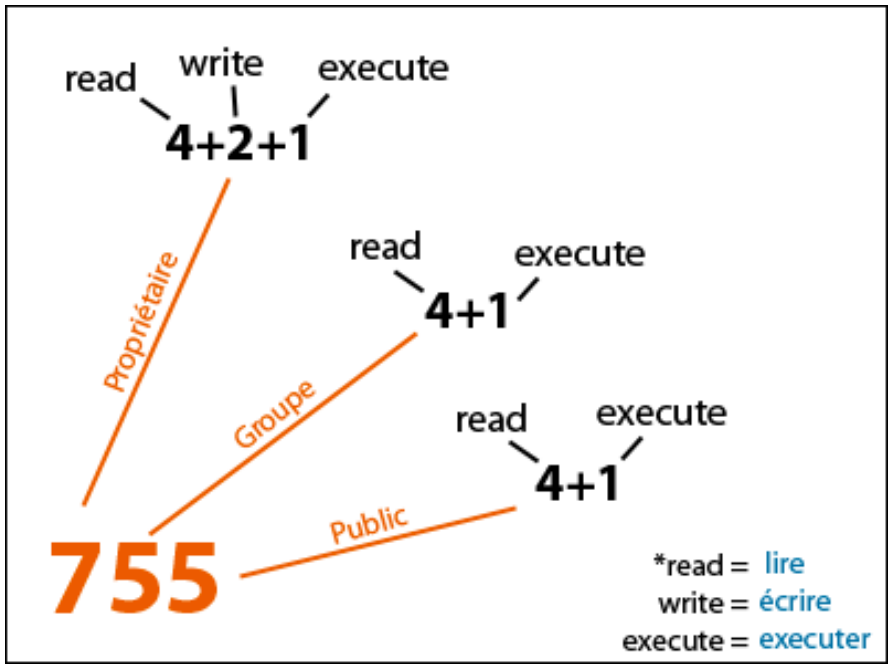
Changement droits d'un fichier - Utilisation des symboles

Exemple :

```
invite/> ls -l toto.py
-rw-r--r-- 1 rouvier staff 486 29 jan 11:57 toto.py
invite/> chmod a+rwx toto.py
invite/> ls -l toto.py
-rwxrwxrwx 1 rouvier staff 486 29 jan 11:57 toto.py
```

Changement droits d'un fichier - Utilisation du nombre octal

- **chmod XXX filename**



Changement droits d'un fichier - Utilisation du nombre octal

- chmod XXX filename
- On affecte directement le mode souhaité en octal (base 8)
- Exemples :
  - **644** Lecture/Ecriture (rw) pour le propriétaire, lecture (r) seulement pour le groupe et les autres
  - **766** rwx user, rw group, rw other
  - **750** rwx user, r group, aucun droit other
- Exemple :

```
invite/> ls -l toto.py
-rw-r--r-- 1 rouvier staff 486 29 jan 11:57 toto.py
invite/> chmod 777 toto.py
invite/> ls -l toto.py
-rwxrwxrwx 1 rouvier staff 486 29 jan 11:57 toto.py
```

Section 2

Manipulation des fichiers

Commandes sur les fichiers

- **basename** : le nom sans le préfixe
- **file** : type du fichier
- **cat** : affiche le contenu sans pause
- **head** : première lignes
- **tail** : dernières lignes
- **touch**
  - Si le fichier existe : maj date dernière modif
  - Si le fichier n'existe pas : créer un fichier vide
- **more** : affiche le contenu avec pause

Basename

La commande **basename** permet de renvoyer le préfixe du fichier placé en paramètre

```
invite/> basename /etc/fichier.sh
fichier.sh
```

```
invite/> basename /etc/fichier.sh .sh
fichier
```

File

La commande **file** permet de déterminer le type d'un fichier :

```
invite/> file fichier.gz
fichier.gz: gzip compressed data, from Unix, max compression
```

```
invite/> file 64x64x32.png
64x64x32.png: PNG image, 64 x 64, 8-bit/color RGBA,
non-interlaced
```

Cat

La commande **cat** permet de visualiser le contenu d'un ou de plusieurs fichiers sur le terminal :

```
invite/> cat fichier.txt
bla, bla, bla !!!!
XXXXX
```

```
invite/> cat fichier1.txt fichier2.txt
bla, bla, bla !!!!
XXXXX
bli, bli, bli !!!!
YYYY
```

Head

La commande **head** permet d'afficher les premières lignes de texte d'un fichier ou de l'entrée standard. :

```
invite/> head fichier.txt
bla1, bla1, bla1 !!!!
bla2, bla2, bla2 !!!!
bla3, bla3, bla3 !!!!
bla4, bla4, bla4 !!!!
```

```
invite/> head -n 1 fichier2.txt
bla1, bla1, bla1 !!!!
```

Tail

La commande **tail** permet d'afficher les dernières lignes de texte d'un fichier ou de l'entrée standard :

```
invite/> tail fichier.txt
bla1, bla1, bla1 !!!!
bla2, bla2, bla2 !!!!
bla3, bla3, bla3 !!!!
bla4, bla4, bla4 !!!!
```

```
invite/> tail -n 1 fichier2.txt
bla4, bla4, bla4 !!!!
```

```
invite/> tail -n +2 fichier2.txt
bla2, bla2, bla2 !!!!
bla3, bla3, bla3 !!!!
bla4, bla4, bla4 !!!!
```

Touch

La commande **touch** permet permettant de modifier le timestamp de dernier accès et de dernière modification d'un fichier. Cette commande permet également de créer un fichier vide :

```
invite/> touch fichier.txt
```

```
invite/> ls -lrth fichier.txt
-rw-r--r-- 1 rouvier staff 486 21 jan 11:57 toto.py
invite/> touch fichier.txt
-rw-r--r-- 1 rouvier staff 486 28 jan 13:00 toto.py
```

More

La commande **more** permet de voir (mais pas de modifier) le contenu d'un fichier texte, page par page :

```
invite/> more fichier.txt
```

Commandes

- Quelques commandes/concepts intéressants
  - **cut** : permet d'afficher des zones spécifiques d'un fichier
  - **sort** : permet de trier les lignes d'un fichier
  - **uniq** : permet d'éliminer les lignes dupliquées dans un fichier trié
  - **wc** : permet de décompter des mots/lignes/caractères
  - **paste** : permet de concaténer les lignes de même niveau des fichiers passés en argument
  - **join** : permet de fusionner les lignes de deux fichiers ayant un champ commun
  - **sed** : permet de transformer ou de supprimer des caractères d'un fichier
  - **find** : permet de chercher des fichiers, et éventuellement d'exécuter une action dessus
  - **grep** : permet de rechercher dans les fichiers des lignes de texte qui satisfont l'occurence

Exemple

Exemple d'un fichier que nous allons utiliser pour les commandes cut, sort, uniq et wc.

```
invite/> cat fichier.txt
3;tutu;11
1;toto;18
1;toto;18
4;titi;13
2;tata;10
```

Cut - Partie 1

La commande **cut** permet d'afficher des zones spécifiques d'un fichier

```
invite/> cut -f1 -d";" fichier.txt
3
1
1
4
2
```

```
invite/> cut -f1,3 -d";" fichier.txt
3;11
1;18
1;18
4;13
2;10
```

```
invite/> cut -f1-3 -d";" fichier.txt
3;tutu;11
1;toto;18
1;toto;18
4;titi;13
```

Cut - Partie 2

La commande **cut** permet d'afficher des zones spécifiques d'un fichier :

```
invite/> cut -c1 -d";" fichier.txt
3
1
1
4
2
```

Sort

La commande **sort** permet de trier les lignes d'un fichier :

```
invite/> sort -n fichier.txt
1;toto;18
1;toto;18
2;tata;10
3;tutu;11
4;titi;13
```

uniq

La commande **uniq** permet d'afficher les lignes d'un fichier texte en supprimant les multiples occurrences consécutives d'une même ligne :

```
invite/> uniq fichier.txt
3;tutu;11
1;toto;18
4;titi;13
2;tata;10
```

```
invite/> uniq -c fichier.txt
1 3;tutu;11
2 1;toto;18
1 4;titi;13
1 2;tata;10
```

Wc

La commande **wc** permet de compter :

- Le nombre de lignes :

```
invite/> wc -l fichier.txt
5
```

- Le nombre de mots :

```
invite/> wc -w fichier.txt
8
```

- Le nombre de caractères :

```
invite/> wc -c fichier.txt
48
```

Paste – Partie 1

La commande **paste** permet de concatèner les lignes de même niveau des fichiers passés en argument :

```
invite/> cat janvier2012
Alimentation 50.00
Eau 25.00
Electricite 123.50
Loyer 456.90
Assurances 234.00
```

```
invite/> cat fevrier2012
Alimentation 67.00
Eau 34.00
Electricite 156.00
Loyer 456.90
Assurances 225.00
```

Paste – Partie 2

La commande **paste** permet de concatèner les lignes de même niveau des fichiers passés en argument :

```
invite/> paster janvier2012 fevrier2012
Alimentation 50.00 Alimentation 67.00
Eau 25.00 Eau 34.00
Electricite 123.50 Electricite 156.00
Loyer 456.90 Loyer 456.90
Assurances 234.00 Assurances 225.00
```

Join – Partie 1

La commande **join** permet de fusionner les lignes de deux fichiers ayant un champ commun :

```
invite/> cat janvier2012
Alimentation 50.00
Eau 25.00
Electricite 123.50
Loyer 456.90
Assurances 234.00
```

```
invite/> cat fevrier2012
Alimentation 67.00
Eau 34.00
Electricite 156.00
Loyer 456.90
Assurances 225.00
```

Join – Partie 2

La commande **join** permet de fusionner les lignes de deux fichiers ayant un champ commun :

```
invite/> join -1 1 -2 1 -d" " janvier2012 fevrier2012
Alimentation 50.00 67.00
Eau 25.00 34.00
Electricite 123.50 156.00
Loyer 456.90 456.90
Assurances 234.00 225.00
```

Sed

La commande **sed** permet de transformer ou de supprimer des caractères d'un fichier

```
invite/> sed "s/Alimentation/ALIMENTATION/g" janvier2012
ALIMENTATION 50.00 67.00
Eau 25.00 34.00
Electricite 123.50 156.00
Loyer 456.90 456.90
Assurances 234.00 225.00
```

Find - Partie 1

La commande **find** permet de chercher des fichiers, et éventuellement d'exécuter une action dessus :

Pour rechercher un fichier, par son nom :

```
invite/> find <emplacement> -name <nom_du_fichier>
```

Quelques exemples :

```
invite/> find /usr/ -name "trash.svg"
/usr/share/themes/adriend-light/cinnamon/trash.svg
```

Attention, le nom est sensible à la casse. Pour pallier à ce problème on utilise l'option **-iname** :

```
invite/> find /usr/ -iname "trash.svg"
/usr/share/themes/adriend-light/cinnamon/trash.svg
/usr/share/themes/adriend-light/cinnamon/TRASH.svg
```

Find - Partie 2

Utilisation poussée :

- Recherche par type de fichier
  Exemple : chercher les **fichiers** dont le nom contient sm

```
invite/> find /var/log/ -type f -name "*sm*"
/var/log/samba/log.smbd /var/log/samba/smbd.log
/var/log/samba/old/log.smbd-20140223
```

Exemple : chercher les **répertoires** dont le nom contient sm

```
invite/> find /var/log/ -type d -name "*sm*"
/var/log/samba/cores/smbd
```

- Recherche par taille
  Exemple : chercher les fichiers de plus de 10Mo dans /usr/bin

```
invite/> find /usr/bin -size +10M
/usr/bin/fgfs /usr/bin/inkscape /usr/bin/clementine
```

Find - Partie 3

Utilisation poussée :

- Recherche par date d'accès
  Exemple : chercher les fichiers dans /var/log/samba qui ont été accédés il y a moins de 1 jour (donc aujourd'hui) :

```
invite/> find /var/log/samba/ -atime -1
/var/log/samba/ /var/log/samba/10.6.0.38.log
/var/log/samba/10.5.26.125.log
```

Exemple : chercher les fichiers dans /var/log/samba qui ont été accédés il y a plus de 90 jours :

```
invite/> find /var/log/samba/ -atime +90
/var/log/samba/192.168.1.30.log /var/log/samba/calvin.log
```

Find - Partie 4

Utilisation poussée :

- Appeler une commande

  Exemple : chercher toutes les images et je veux effectuer un chown pour changer son utilisateur

```
invite/> find / -name "*.jpg" -exec chown adrien {} \;
```

  - **{}** : représente le nom du fichier trouvé
  - \; : représente la fin de la commande

  Exemple : copier toutes les fichiers textes dans un autre répertoire :

```
invite/> find /home/david/ -name "*.txt" -exec cp {}
/home/david/toto/ \;
```

Grep

La commande **grep** permet de rechercher dans les fichiers des lignes de texte qui satisfont l'occurence
Exemple : chercher un motif dans un fichier

```
invite/> grep "chaine" fichier.txt
```

Exemple : chercher un motif dans un ensemble de fichier

```
invite/> grep "chaine" *.txt
```

Section 3

Historique des commandes

Historique des commande - Partie 1

La commande **history** permet de voir rapidement les dernières commandes exécutées et ainsi pouvoir les re-exécutés.

- **Afficher l'historique**

```
invite/> history
1 mkdir toto
2 exit
3 ls -la
4 pwd
```

- **Afficher les *n* dernières lignes**

```
invite/> history 3
2 exit
3 ls -la
4 pwd
```

Historique des commande - Partie 2

- **Répéter la dernière commande**

```
invite/> date
Dim 20 jan 2019 21:01:29 CET
invite/> !!
date
Dim 20 jan 2019 21:01:31 CET
```

- **Répéter une commande spécifique**

```
invite/> history 2
101 date
102 history 2
invite/> !101
date
Dim 20 jan 2019 21:01:35 CET
```

Historique des commande - Partie 3

- **Répéter une commande spécifique**

```
invite/> history
101 date
102 history 2
invite/> !-2
date
Dim 20 jan 2019 21:01:35 CET
```

- **Répéter des commandes précédentes**

En utilisant les touches haut et bas du clavier.

Historique des commande - Partie 4

- **Répéter une commande qui commence par une chaîne de caractère**

```
invite/> systemctl start httpd
invite/> systemctl stop chronyd
invite/> systemctl restart chronyd
invite/> !systemctl
restart chronyd
```

Attention : cela peut être dangereux de relancer la dernière commande si elle est différente de ce que vous attendiez.

```
invite/> !systemctl:p
systemctl restart chronyd
```

Historique des commande - Partie 5

- **Rechercher dans l'historique**

On peut rechercher une commande dans l'historique en utilisant les touches Control+R.

```
invite/> Appuyer sur les touches Control+R, cela permettra
d'afficher le prompt reverse-i-search
(reverse-i-search)`red': cat /etc/redhat-release
```

- **Substituer une commande de l'historique tout en gardant les paramètres**

```
invite/> ls anaconda-ks.cfg
anaconda-ks.cfg
invite/> vi !!:$
```

Historique des commande - Partie 6

- **Substituer une commande de l'historique tout en gardant un paramètre**

```
invite/> cp anaconda-ks.cfg anaconda-ks.cfg.bak
invite/> vi !!:1
vi anaconda-ks.cfg
```

```
invite/> cp anaconda-ks.cfg anaconda-ks.cfg.bak
invite/> vi !!:2
vi anaconda-ks.cfg.bak
```

- **Supprimer l'historique**

```
invite/> history -c
```

Section 4

Expressions réguliéres

Introduction

- **Problématique** : Comment rechercher une chaîne de caractère
- Une expression régulière est une chaîne de caractère (motif, pattern) qui décrit selon une syntaxe précise un ensemble de chaîne de caractères précises.
- Avec les expressions régulières, vous pouvez :
  - rechercher des fichiers
  - rechercher une adresse email dans un fichier
  - vérifier si une adresse email est correcte

Exemple, pour rechercher toutes les adresse emails dans fichier.txt via la commande egrep :

```
invite/> egrep "[a-z.]*@[a-z-]*.[a-z]*" fichier.txt
```

Recherche d'un pattern

Recherche d'un pattern dans un texte :

Exemple	Matches	Not Match
petit	le petit chaperon rouge	le grand chaperon rouge

Exemple :

```
invite/> egrep "petit" fichier.txt
```

Quantificateur

Permet de compter le nombre de fois qu'une lettre peut apparaître

- a ? : a peut apparaitre 0 ou 1 fois
- a+ : a peut apparaitre au moins 1 fois
- a* : a peut apparaitre 0, 1 ou plusieurs fois
- a{n} : a doit apparaitre n fois
- a{n,m} : a doit apparaitre de n à m fois
- a{n,} : a doit apparaitre au moins n fois
- . : n'importe quel caractère

Exemple	Matches	Not Match
bor ?is	boris, bois	booris
o+h	oh, ooooh	oohhhhhh
peti*t	pett, petit, petiiiiit	petitttt
pet.t	petit, petat, petet	pett
9{3}	999	9, 99, 9999
9{1,3}	9,99,999	99999
9{2,}	99,999,99999	9

Exemple :

```
invite> egrep "bor?is" fichier.txt
```

Alterner

Permet de compter le nombre de fois qu'une lettre peut apparaître

- | : recherche quelque chose ou quelque chose

Exemple	Matches	Not Match
chien|chat	chien, chat	oiseau

Exemple :

```
invite/> egrep "chien|chat" fichier.txt
```

Classe de caractère

Une classe de caractère permet de rechercher parmi plusieurs jeu de caractère, simplement en mettant les jeux de caractère entre crochet

- [az] : un seul caractère donné dans les backets
- [a-z] : un caractère de la plage indiqué dans les brackets

Exemple	Matches	Not Match
gr[ae]y	gray, grey	grry
gr[a-z]y	gray, grey	grry
gr[0-9]y	gr0y, gr9y	gray
gr[a-z0-9]y	gray, grey, gr0y, gr9y	gr-y

Exemple :

```
invite/> egrep "gr[a-z]y" fichier.txt
```

Ancrage

Une classe de caractère permet de rechercher parmi plusieurs jeu de caractère, simplement en mettant les jeux de caractère entre crochet

- ^ : début de ligne
- $: fin de ligne

Exemple :

```
invite/> egrep "^grey" fichier.txt
```