THE INNOVATIVE UNIVERSITY

# Commandes Linux: grep, regex et sed

**Ilias TOUGUI**

L'Ecole Supérieure d'Informatique et du Numérique

INF2132 - Systèmes d'Exploitation

# PLAN DU COURS

1. La commande: grep
2. Les expressions régulières
3. La commande: sed

# La commande grep

# La commande grep

**Syntaxe**:

**grep** *[options] "motif" fichier(s)*

**Description**:

La commande **grep** permet de rechercher du text et des expressions régulières dans des fichiers. Elle affiche les noms de fichiers ainsi que les lignes contenant l'expression.

# La commande grep - suite

**Syntaxe**:

**grep** *[options] "motif" fichier(s)*

**Exemple**:

```
# Rechercher le mot "error" dans un fichier de log
```

```
$ grep "error" /var/log/syslog
```

# La commande grep - suite

**Syntaxe**:

**grep** *[options] "motif" fichier(s)*

**Options courantes:**

- **-i**: ne tient pas compte des minuscules et des MAJUSCULES

**Exemple**:

```
# Trouve "error", "Error", "ERROR", etc.
```

```
$ grep -i "error" /var/log/syslog
```

# La commande grep - suite

**Syntaxe**:

**grep** *[options] "motif" fichier(s)*

**Options courantes:**

- **-c**: compte les occurrences d'un motif (les lignes)

**Exemple**:

```
# Affiche le nombre de lignes contenant "error"
```

```
$ grep -c "error" /var/log/syslog
$ 100183
```

# La commande grep - suite

**Syntaxe**:

**grep** *[options] "motif" fichier(s)*

**Options courantes:**

- **-n**: afficher les numéros de lignes contenant un motif

**Exemple**:

```
# Affiche le numero de la ligne contenant "11.773203+01"
```

```
$ grep -n "11.773203+01" /var/log/syslog
$ 107476 ...
```

# La commande grep - suite

**Syntaxe**:

**grep** *[options] "motif" fichier(s)*

**Options courantes:**

- **-r**: Lit tous les fichiers sous chaque répertoire, de manière récursive

**Exemple**:

```
# Recherche dans tous les fichiers du répertoire
```

```
$ grep -r "fatalErrorCount" /home/$USER/
```

# Quiz 1: Commande grep - Bases

**Question 1.1 - Syntaxe de base**:

*Quelle commande permet de rechercher le mot "kernel" dans le fichier /var/log/syslog en ignorant la casse ?*:

1. grep kernel /var/log/syslog
2. grep -i "kernel" /var/log/syslog
3. grep -c "kernel" /var/log/syslog
4. grep -n "kernel" /var/log/syslog

# Quiz 1: Commande grep - Bases

**Question 1.1 - Syntaxe de base**:

*Quelle commande permet de rechercher le mot "kernel" dans le fichier /var/log/syslog en ignorant la casse ?*:

**Réponse:**

```
2.  L'option -i rend la recherche insensible à la casse
$ grep -i "kernel" /var/log/syslog
```

# Quiz 1: Commande grep - Bases

**Question 1.2 - Options grep**:

*Que fait la commande:* ***grep -n "error" /var/log/syslog*** *?*

1. Affiche les lignes avec leurs numéros de ligne
2. Compte les lignes contenant "error"
3. Recherche récursivement dans les sous-répertoires
4. Ignore la casse

# Quiz 1: Commande grep - Bases

**Question 1.2 - Options grep**:

*Que fait la commande:* ***grep -n "error" /var/log/syslog*** *?*

**Réponse:**

```
1.  L'option -n affiche les lignes avec numéros de lignes
```

# Quiz 1: Commande grep - Bases

**Question 1.3 - Pratique**:

*Comment afficher uniquement le nombre de lignes contenant **"ssh"** dans **/var/log/auth.log** ?*

1. grep -l "ssh" /var/log/auth.log
2. grep -v "ssh" /var/log/auth.log
3. grep -c "ssh" /var/log/auth.log
4. grep -r "ssh" /var/log/auth.log

# Quiz 1: Commande grep - Bases

**Question 1.3 - Pratique**:

*Comment afficher uniquement le nombre de lignes contenant **"ssh"** dans **/var/log/auth.log** ?*

**Réponse:**

```
3.  L'option -c compte les lignes correspondantes
$ grep -c "ssh" /var/log/auth.log
```

# Les expressions régulières regex

# Introduction aux Expressions Régulières

Les expressions régulières (**regex**) sont un langage de motifs permettant de rechercher, valider et manipuler du texte de façon très puissante. Elles représentent des **séquences de caractères** qui définissent un modèle de recherche.

Pourquoi Apprendre les Regex:

- **Efficacité**: Une ligne de regex peut remplacer des dizaines de lignes de code.
- **Universalité**: Supportées dans la plupart des langages (Python, JavaScript, PHP) et outils (grep, sed, awk).
- **Puissance**: Permettent des recherches complexes impossibles autrement.

# Regex - Le Métacaractère Point (.)

**Définition:**
Le point **.** représente n'importe quel caractère (sauf retour de ligne)

**Exemples:**

```
# Chercher "wl" suivi de n'importe quel caractère (wlan0, wlp1, etc.)
$ grep "wl." /var/log/dmesg
```

```
# Chercher "PID" suivi de 4 caractères quelconques
$ grep "PID...." /var/log/syslog
```

Pourquoi c'est utile?

- Les numéros changent (wlw, wlp, wl-...)
- Le point permet de trouver toutes les variantes en une seule fois !

# Ancres (^ et $) - Début et Fin

**Définition:** Ces symboles indiquent où dans la ligne doit se trouver votre motif.

- ^= "commence par"
- $ = "finit par"

**Exemples:**

```
# Chercher les logs générés le 2025-09-22
$ grep "^2025-09-22" /var/log/syslog
```

```
# Chercher les logs qui finis par "users."
$ grep "users.$" /var/log/syslog
```

```
# Combiner les deux lignes
$ grep "^2025-09-22" /var/log/syslog | grep "users.$"
```

# Les Intervalles

Voici quelques exemples d'intervalles:

| Intervalle | Equivalent | Traduction |
|---|---|---|
| [a-z] | [abcdefghijklmonp qrstuvwxyz] | Lettres minuscules de a à z |
| [A-Z] | [ABCDEFGHIJKLMNOP QRSTUVWXYZ] | Lettres majuscules de A à Z |
| [0-9] | [0123456789] | Chiffres de 0 à 9 |
| [a-z0-9] | [abcdefghijklmonp qrstuvwxyz0123456789] | Lettres minuscules de a à z ou chiffres |

# Classes de Caractères [...] - Choix Multiples

**Définition:** Les crochets permettent de dire : "n'importe lequel de ces caractères": [abc] = "accepte a OU b OU c"

**Exemples:**

```
# Chercher "loop" suivi de 0 OU 1 OU 2
$ grep "loop[012]" /var/log/dmesg
```

```
# Chercher les disques sda, sdb, sdc, sdd...
$ grep "sd[abcd]" /var/log/syslog
```

```
# [a-z] = n'importe quelle lettre minuscule
$ grep "python3-[a-z]" /var/log/dpkg.log
```

```
# [A-Z] = n'importe quelle lettre majuscule
$ grep "[A-Z]rror" /var/log/syslog
```

# Classes Négatives [^...] - Tout Sauf

**Définition:** Le ^ dans les crochets signifie "PAS" ou "SAUF": [^abc] = "tout caractère SAUF a, b, ou c"

**Exemples:**

```
# Chercher "loop" suivi de tout caractère sauf 0 OU 1 OU 2
$ grep "loop[^012]" /var/log/dmesg
```

```
# Chercher "socket-" suivi de tout caractère sauf a OU b OU c
$ grep "socket-[^abcd]" /var/log/syslog
```

# Les Quantificateurs

Les quantificateurs en expressions régulières permettent de spécifier le nombre exact ou approximatif d'occurrences d'un élément dans un pattern.

1. **{min,max}**: Le quantificateur {min,max} spécifie que l'élément précédent doit apparaître entre **min** et **max** fois inclusivement.

**Exemples:**

```
# Trouve les adresses IP où chaque octet contient entre 1 et 3 chiffres (ex: 192.168.1.1, 10.0.0.254)
$ grep -E "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}" /var/log/syslog
```

# Les Quantificateurs - Suite

```
$ grep -E "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}" /var/log/syslog
```

**Remarque:**
Cette commande utilise des expressions régulières de base (le comportement par défaut de grep). En mode BRE, les caractères {, }, +, ?, |, (, et ) sont traités comme des caractères littéraux plutôt que comme des métacaractères. Cela signifie que grep recherche la chaîne littérale contenant des accolades, en recherchant des motifs tels que [0-9]{1,3} au lieu d'interpréter les accolades comme des quantificateurs.

**Solution avec grep simple:**

```
$ grep "[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}" /var/log/syslog
```

# Les Quantificateurs - Suite

2. **{min,}**: Le quantificateur {min,} indique que l'élément doit apparaître au minimum min fois, sans limite maximale.

**Exemples:**

```
# Trouve les lignes contenant "error" suivi d'au moins 10 caractères supplémentaires pour capturer les messages d'erreur complets
$ grep -E "error.{10,}" /var/log/syslog
```

```
# Trouve tous les mots d'au moins 8 caractères, utile pour identifier des noms de services ou fichiers spécifiques
$ grep -E "[a-zA-Z]{8,}" /var/log/syslog
```

# Les Quantificateurs - Suite

3. **{,max}**: Le quantificateur ,max permet entre 0 et max occurrences de l'élément précédent.

**Exemples:**

```
# Trouve "error" suivi d'un espace et d'un code numérique de 0 à 3 chiffres
(ex:  "error 404", "error 1", "error")
$ grep -E "error [0-9]{,3}" /var/log/syslog
```

```
# Trouve le log système qui commence par une date
$ grep -E "^[0-9]{,4}" /var/log/syslog
```

# Les Quantificateurs - Suite

4. **{nombre}**: Le quantificateur {nombre} exige exactement le nombre spécifié d'occurrences de l'élément précédent.

**Exemples:**

```
# Trouve les adresses MAC avec exactement 2 caractères hexadécimaux par segment (ex: 00:1B:44:11:3A:B7)
$ grep -E "([0-9a-fA-F]{2}:){5}[0-9a-fA-F]{2}" /var/log/syslog
```

Cette expression régulière est conçue pour détecter des adresses MAC dans les logs système.

- '([0-9a-fA-F]{2}:){5}' = 5 premières paires hexadécimales avec deux-points
- '[0-9a-fA-F]{2}' = dernière paire hexadécimale sans deux-points

# grep, grep -E et egrep

- **grep** utilise les expressions régulières de base (BRE) par défaut. Dans ce mode, lesde, les métacaractères comme +, ?, {, }, |, (, ) doivent être échappés avec des **backslashes** pour fonctionner comme métacaractères
- **grep -E** et **egrep** sont fonctionnellement identiques. La commande **egrep** est maintenue pour la compatibilité historique, mais **grep -E** est la syntaxe recommandée dans les systèmes modernes

| Commande | Type regex | Métacaractères | Exemple |
|---|---|---|---|
| grep | BRE | Échappés {, \+ | grep "test\{2,3\}" |
| grep -E | ERE | Directs {, + | grep -E "test{2,3}" |
| egrep | ERE | Directs {, + | egrep "test{2,3}" |

# Quiz 2: Expressions régulières avec grep

**Question 2.1 - Métacaractères**:

*Que recherche **grep "wl." /var/log/dmesg** ?*

1. Le mot "wl" suivi d'un point littéral
2. "wl" en fin de ligne
3. "wl" en début de ligne
4. "wl" suivi de n'importe quel caractère

# Quiz 2: Expressions régulières avec grep

**Question 2.1 - Métacaractères**:

*Que recherche **grep "wl." /var/log/dmesg** ?*

**Réponse:**

```
4.  Le point représente n'importe quel caractère
```

# Quiz 2: Expressions régulières avec grep

**Question 2.2 - Ancres**:

*Quelle commande trouve les lignes qui se terminent par **"users."** ?*

1. grep "users.$" /var/log/syslog
2. grep "^users." /var/log/syslog
3. grep "users." /var/log/syslog
4. grep ".users" /var/log/syslog

# Quiz 2: Expressions régulières avec grep

**Question 2.2 - Ancres**:

*Quelle commande trouve les lignes qui se terminent par **"users."** ?*

**Réponse:**

1. `$ indique la fin de ligne, et .  correspond à n'importe quel caractère`

# Quiz 2: Expressions régulières avec grep

**Question 2.3 - Classes de caractères**:

*Que fait **grep "loop" /var/log/dmesg** ?*

1. Recherche "loop012"
2. Recherche "loop0", "loop1", ou "loop2" ...
3. Recherche "loop" suivi de 0 à 12 caractères
4. Erreur de syntaxe

# Quiz 2: Expressions régulières avec grep

**Question 2.3 - Classes de caractères**:

*Que fait **grep "loop" /var/log/dmesg** ?*

**Réponse:**

```
2.  Recherche "loop0", "loop1", ou "loop2" ...
# Les crochets définissent un ensemble de caractères possibles
```

# La commande sed

# La commande sed

**Syntaxe**:

**sed** *[options] "script" fichier(s)*

**Description**:

La commande **sed** (Stream Editor) est un éditeur de flux qui permet de modifier le contenu de fichiers texte de manière automatisée sans ouvrir le fichier dans un éditeur traditionnel.

**Options principales**:

- **-i**: modification directe du fichier (in-place)
- **-n**: supprime l'affichage automatique
- **-e**: permet d'exécuter plusieurs commandes

# La commande sed - Suite

**Exemples**:

```
# Optionnel:  copier le fichier syslog pour le modifier
$ cp /var/log/syslog ~/systemlog
```

```
# Remplacer la première occurrence de "error" dans chaque ligne par "erreur"
(affiche le resultat sans modification du fichier original)
$ sed "s/error/erreur/" systemlog
```

```
# Remplacer la première occurrence de "error" par "erreur" (avec modification
du fichier original)
$ sed -i "s/error/erreur/" systemlog
```

# La commande sed - Suite

**Syntaxe**:

**sed** *[options] "script" fichier(s)*

**Flags principaux**:

- **g**: remplacement global (toutes les occurrences)
- **i**: insensible à la casse
- **p**: affiche les lignes modifiées
- **nombre**: remplace la nième occurrence

**Exemple**:

```
# Remplacer toutes les occurrences (flag g)
$ sed "s/error/erreur/g" bsyslog
```

# La commande sed - Suite

**Exemple**:

```
# Remplacement global insensible à la casse
```

```
$ sed -i "s/ERROR/erreur/gi" systemlog
$ grep "ERROR" systemlog
$
```

```
# Remplace la deuxieme occurence de 'keysym' dans la ligne
```

```
$ grep "keysym" systemlog
$ sed -i "s/keysym/KEYSYM/2" systemlog
$ grep "KEYSYM" systemlog
```

# La commande sed - Suite

**Syntaxe**:

**sed** 'numéro_ligne s/motif/remplacement/' fichier

**Exemple**:

```
# Modifier seulement la ligne 5
$ head systemlog
$ sed -i "5s/kernel/KERNEL/" systemlog
```

# La commande sed - Suite

**Syntaxe**:

**sed** ’début,fin s/motif/remplacement/’ fichier

**Exemple**:

```
# Modifier les lignes 5 à 7 qui contient le mot ’Bluetooth’
$ sed -i "5,7s/Bluetooth/BLE/g" systemlog
```

```
# Modifier depuis la ligne 10 jusqu’à la fin
$ head systemlog
$ sed -i "10,$s/old/new/g" systemlog
```

# La commande sed - Suite

**Commande de suppression**:

```
# Supprimer la ligne 3
$ sed -i "3d" systemlog
```

```
# Supprimer les lignes 2 à 5
$ sed -i "2,5d" fsystemlog
```

```
# Supprimer les lignes contenant "error"
$ sed -i "/error/d" systemlog
```

```
# Supprimer les lignes vides
$ sed -i "/^$/d" systemlog
```

# La commande sed - Insertion et ajout

**Commande d'insertion et d'ajout**:

**Commandes**:

- **i**: insérer avant la ligne
- **a**: ajouter après la ligne

**Exemples:**

```
# Insérer une ligne avant la ligne 2
$ sed -i "2i\Nouvelle ligne insérée" systemlog
```

```
# Ajouter une ligne après chaque ligne contenant "config"
$ sed -i "/config/a Log ajoutée apres config" systemlog
```

# Quiz 3: Commande sed - Substitution

**Question 3.1 - Syntaxe de base**:

*Que fait **sed "s/ancien/nouveau/" fichier.txt** ?*

1. Supprime les lignes contenant "ancien"
2. Compte les occurrences de "ancien"
3. Remplace toutes les occurrences de "ancien" par "nouveau"
4. Remplace la première occurrence de "ancien" par "nouveau" sur chaque ligne

# Quiz 3: Commande sed - Substitution

**Question 3.1 - Syntaxe de base**:

*Que fait **sed "s/ancien/nouveau/" fichier.txt** ?*

**Réponse:**

```
4.  Remplace la première occurrence de "ancien" par "nouveau" sur chaque
ligne
# Sans le flag "g", sed ne remplace que la première occurrence par ligne.
```

# Quiz 3: Commande sed - Substitution

**Question 3.2 - Flag global**:

*Comment remplacer TOUTES les occurrences de **"test"** par **"TEST"** dans un fichier ?*

1. sed "s/test/TEST/" fichier.log
2. sed "s/test/TEST/g" fichier.log
3. sed "s/test/TEST/a" fichier.log
4. sed "g/test/TEST/s" fichier.log

# Quiz 3: Commande sed - Substitution

**Question 3.2 - Flag global**:

*Comment remplacer TOUTES les occurrences de **"test"** par **"TEST"** dans un fichier ?*

**Réponse:**

```
$ sed "s/test/TEST/g" fichier.log
```

2. Le flag 'g' active le remplacement global sur chaque ligne

# Quiz 3: Commande sed - Substitution

**Question 3.3 - Modification directe**:

*Quelle option permet de modifier directement le fichier avec sed ?*

1. -d
2. -m
3. -i
4. -f

# Quiz 3: Commande sed - Substitution

**Question 3.3 - Modification directe**:

*Quelle option permet de modifier directement le fichier avec sed ?*

**Réponse:**

```
3. L'option -i modifie le fichier directement (in-place)
```

```
$ sed -i "s/test/TEST/" fichier.log
```