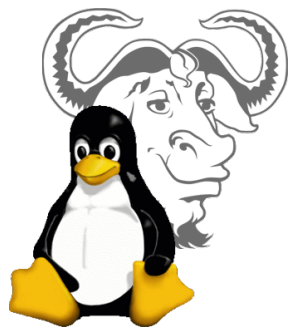
# Administration d'un système GNU / Linux

## 04 — grep, sed, expressions régulières

Anthony Labarre

上海师范大学

23 octobre 2024



# Plan d'aujourd'hui

1. `grep`
2. `sed`
3. Expressions régulières

# grep

# grep

- `grep` est l'un des programmes les plus utiles de GNU;
- Il permet de récupérer toutes les lignes d'un texte contenant ou évitant un morceau de texte donné (= "motif");
- Il permet aussi d'utiliser des **expressions régulières** (plus tard);

`grep` est un filtre: on peut l'utiliser sur des fichiers ou avec des *pipes*.

# grep: premier exemple

## Syntaxe de grep

```
grep [options] motif fichier(s)
```

Par exemple:

- `grep bonjour fichier.txt;`
- `grep "au revoir" fichier1.txt fichier2.txt;`
- `grep -c 老师 employés_shnu.txt;`
- ...

Examinons ensemble quelques exemples.

# grep: premier exemple

Utilisation la plus simple: chercher toutes les occurrences exactes d'un mot (en respectant la casse).

## Exemple

```
$ grep Quasimodo Victor\ Hugo\ -\ Notre-Dame\ de\ Paris.txt
# affiche toutes les lignes contenant Quasimodo
```

```
$ grep -c Quasimodo Victor\ Hugo\ -\ Notre-Dame\ de\ Paris.txt
241 # affiche le nombre de lignes contenant Quasimodo
```

```
$ grep -o Quasimodo Victor\ Hugo\ -\ Notre-Dame\ de\ Paris.txt
# affiche chaque occurrence sur une ligne séparée
Quasimodo
Quasimodo
...
```

```
$ grep -o Quasimodo Victor\ Hugo\ -\ Notre-Dame\ de\ Paris.txt | wc -l
244 # le nombre d'occurrences de Quasimodo dans le livre
```

# grep: deuxième exemple

On peut demander à ignorer la casse:

## Exemple

```
$ grep -c Quatre Victor\ Hugo\ -\ Notre-Dame\ de\ Paris.txt
9 # le nombre d'occurrences de "Quatre"
```

```
$ grep -c -i Quatre Victor\ Hugo\ -\ Notre-Dame\ de\ Paris.txt
120 # le nombre d'occurrences de "Quatre" et "quatre" et ...
```

Attention: grep ne considère pas “Quatre” comme un mot isolé!

Parmi les occurrences, on retrouvera donc aussi “Quatrelivres”, “Quatre-Couronnes”, “Quatre-Nations”, . . .

# Variantes

Il existe quelques variantes utiles de `grep`:

- `pgrep`: recherche un motif parmi les noms des processus;
- `agrep`: recherche de motifs avec des “erreurs”;
  - exemple:
    ```
    agrep -1 "donjon" Victor\ Hugo\ -\ Notre-Dame\ de\
    Paris.txt
    ```
- `ngrep`: recherche de motifs dans le trafic réseau;
- `pdfgrep`: `grep` pour les fichiers pdf;
- `zipgrep`: `grep` pour les fichiers zip . . . sans devoir les décompresser!
- . . .

# sed

# sed

- `grep` est très utile pour trouver des motifs dans un texte . . .
- . . . mais comment faire si l'on veut *remplacer* des motifs par d'autres motifs?
- On utilisera un autre programme: `sed` (**s**tream **ed**itor);

`sed` est aussi un filtre: on peut l'utiliser sur des fichiers ou avec des *pipes*.

# Commandes de sed

- sed remplace des motifs par d'autres dans n'importe quel flux — donc des fichiers, mais aussi des sorties de *pipes*;
- Par défaut, sed écrit son résultat sur STDOUT;
- Il possède de nombreuses commandes, mais nous n'en examinerons que deux:
  1. s: remplacement de texte;
  2. y: remplacement de caractères;

# Remplacement de texte avec sed

## Syntaxe de sed (commande s)

`sed s/avant/après/g fichier`

remplace toutes les occurrences de “avant” par “après” dans `fichier` (la casse est respectée).

“avant” peut être une chaîne ou une expression régulière (voir plus loin). Il y a beaucoup d’autres options que g.

*Examinons quelques exemples.*

# Remplacement de caractères avec sed

On peut aussi remplacer certains caractères par d'autres:

## Syntaxe de sed (commande y)

```
sed y/chaine_1/chaine_2/ fichier
```

remplace `chaine_1[0]` par `chaine_2[0]`, `chaine_1[1]` par `chaine_2[1]`, etc.

Pour la commande `y`, `chaine_1` et `chaine_2` doivent avoir la même longueur!

*Examinons quelques exemples.*

# Entrée et sortie de sed

- Par défaut, sed lit les données d'un fichier;
- Vous pouvez tester sed sur des chaînes avec `echo chaine | sed ...`;

- Par défaut, sed écrit sur STDOUT;
- Si vous voulez écrire le résultat dans un fichier:
  1. utilisez `sed ... entree > sortie` si `entree` $\neq$ `sortie`;
  2. ou `sed -i ... entree` si vous voulez écrire directement les changements dans `entree`;

Si vous débutez avec sed, vérifiez son résultat avant d'utiliser `-i`!

# Expressions régulières

# Expressions régulières

- Une **expression régulière** est une chaîne qui encode un *modèle* de texte;
- Exemples:
  - les mots en minuscules;
  - LES MOTS EN MAJUSCULES;
  - Les Mots Qui Commencent Par Une Majuscule;
  - les numéros de téléphone;
  - les adresses e-mail;

  ...
- Utilité: au lieu de chercher un mot précis dans un texte, on peut chercher tous les morceaux de texte respectant une certaine structure;

# Applications

Recherchons toutes les dates “avant Jésus-Christ” dans le texte suivant (par exemple: 1300 av. J.-C. = 1300 B.C. = 前1300年):

# Ressources

Les expressions régulières peuvent parfois être difficiles à interpréter.

Le site https://regexr.com/ vous permet de tester en direct vos expressions régulières, avec le texte de votre choix.

N'hésitez pas à utiliser cette ressource pour vous entraîner et vérifier vos expressions.

# Premier exemple: les dates

- Construisons une expression permettant de décrire les dates au format international (`YYYY-MM-DD`, par exemple: 2025-10-08);
  - l'année est `YYYY` $\in [\![0, 9999]\!]$, donc quatre entiers dans `[0, 9]`;
  - le mois est `MM` $\in [\![1, 12]\!]$, donc un entier dans `[0, 1]` suivi d'un entier dans `[0, 9]`;
  - le jour est `DD` $\in [\![1, 31]\!]$, donc un entier dans `[0, 3]` suivi d'un entier dans `[0, 9]`;
- Le résultat est l'expression régulière suivante:

```
[0-9][0-9][0-9][0-9]-[0-1][0-9]-[0-3][0-9]
(année)             (mois)      (jour)
```

- Si on préfère les dates chinoises:

```
[0-9][0-9][0-9][0-9]年[0-1][0-9]月[0-3][0-9]日
```

# Remarques sur les dates

- Notre expression ne distingue pas les dates justes des dates fausses:
  - elle accepte des mois faux: 00, ou des nombres $> 12$;
  - elle accepte des jours faux: 00, ou des nombres $> 31$;
- On n'essaiera pas de la corriger, car une expression correcte deviendrait très compliquée:
  - certains mois ont 30 ou 31 jours;
  - le mois de février a parfois 28 jours, parfois 29;

- On préférera récupérer toutes les dates en supposant qu'elles sont correctes, puis éventuellement les filtrer autrement;

# Deuxième exemple: les prénoms (européens)

- Un prénom commence par une majuscule, suivie de n'importe quel nombre de lettres minuscules;
  - la première lettre est donc n'importe quelle majuscule dans `[A, Z]`;
  - les lettres suivantes sont n'importe quelle minuscule dans `[a, z]` (oublions les caractères spéciaux pour simplifier);
  - l'opérateur `*` dans l'expression `x*` précise que `x` apparaît n'importe quel nombre de fois (éventuellement 0);
- Le résultat est l'expression régulière suivante:

```
[A-Z][a-z]*
```

...qui trouvera Marie, Yves, Louis, ... mais pas Jean-Paul (pourquoi?)

# Les bases

Les **éléments** d'une expression régulière peuvent être:

- des caractères "normaux": a, b, c, 1, 2, A, Z, é, è, 九, ...;
- des caractères "spéciaux": `\t`, `\n`, ...;
- des ensembles de caractères: `[az]` = "a ou z";
- des intervalles: `[a-z]`, `[A-Z]`, `[0-9]`, ...;

| Symboles fréquents | | Négations | |
|---|---|---|---|
| `[ab]` | a ou b | `[^ab]` | tout sauf a et b |
| `[0-9]` ou `\d` | les chiffres | `\D` | tout sauf un chiffre |
| `[a-z]` | les lettres minuscules | | |
| `[A-Z]` | les lettres majuscules | | |
| `\w` | les lettres et les chiffres | `\W` | tout sauf une lettre ou un chiffre |
| `\s` | les espaces | `\S` | tout sauf un espace |
| `.` | tout sauf `\n` | | |

# Les répétitions

On indique les répétitions de caractères avec les opérateurs suivants (format: symbole suivi de l'opérateur, sans espace):

- `?`: 0 ou 1 fois;
- `+`: au moins une fois;
- `*`: autant de fois qu'on veut;
- `{n}`: exactement `n` fois;
- `{n,}`: au moins `n` fois;
- `{,m}`: au plus `m` fois;
- `{n,m}`: entre `n` et `m` fois;

# Remarques

- On peut combiner les intervalles: `[a-zA-Z]` = n'importe quel caractère non accentué (minuscule ou majuscule);
- Si vous avez besoin d'un caractère spécial, "échappez"-le; par exemple:
  - `a*` = "n'importe quel nombre de a";
  - `a\*` = "a suivi d'une étoile";

# Expression plus simple pour les dates

On peut maintenant simplifier l'expression qu'on a obtenue pour les dates:

```
[0-9][0-9][0-9][0-9]-[0-1][0-9]-[0-3][0-9]
                    ↓
\d\d\d\d-[0-1]\d-[0-3]\d
                    ↓
\d{4}-[0-1]\d-[0-3]\d
```

Tous les programmes ne connaissent pas les notations "raccourcies" (par exemple, `sed` ne comprend pas `\d`). Pensez à vérifier!

# Exemple plus avancé: les nombres naturels

- L'expression régulière pour les dates autorise la présence de 0 comme "remplissage" (par exemple: 0635 pour l'année 635);
- Comment exprimer les nombres naturels sans remplissage?
- `[1-9][0-9]*` exprime les naturels qui ne commencent pas par 0 ... et on ne peut donc pas représenter 0!
- On utilise l'opérateur logique **or** pour exprimer qu'un naturel est "soit 0, soit un nombre ne commençant pas par 0";
- Cet opérateur s'écrit | (attention: ceci n'est pas un *pipe*);
- Cela donne:

```
0|[1-9][0-9]*
```

# Utilisation des expressions régulières avec grep

- Pour expliquer à grep que le motif est une expression régulière et pas une chaîne “normale”, on utilise l’option -E ou -P;
- grep nous affiche chaque ligne contenant ce qu’on cherche en entier;
- Pour n’afficher que ce qui nous intéresse, on utilise l’option -o: elle affiche:
  - uniquement la partie correspondant à l’expression donnée;
  - chaque résultat sur une ligne différente;

# Place des expressions régulières

On peut aussi préciser **où** les résultats doivent apparaître, avec les opérateurs suivants:

- `^expression`: l'expression doit apparaître au début de la ligne; attention: `[^x]` $\neq$ `^x` !
- `expression$`: l'expression doit apparaître à la fin de la ligne; attention: `[x$]` $\neq$ `x$` !
- `\bexpression`: l'expression doit démarrer un "mot";
- `expression\b`: l'expression doit terminer un "mot";
- `\bexpression\b`: l'expression doit être un "mot";

Examinons quelques exemples ... (voir aussi l'exemple abba sur RegExr.)

# Différents formats

Il existe différentes conventions (ou formats) pour les expressions régulières!

- Testez donc vos expressions, et consultez les manuels des programmes!
- Par exemple, `[0-9]+` et `\d+` sont équivalentes . . . mais:
  - ✓ `grep -E "[0-9]+" fichier` fonctionne;
  - × `grep -E "\d+" fichier` ne fonctionne pas;
  - ✓ `grep -P "\d+" fichier` fonctionne;

# Quelques mots sur les expressions régulières chinoises

- `grep` accepte les caractères chinois, y compris dans les expressions régulières;
- `pcregrep` aussi mais il faut rajouter l'option `-u`;
- Les intervalles ne fonctionnent pas: au lieu de `[一-十]`, on est obligé (?) d'écrire `[一二三四五六七八九十]`;
- `"\p{Han}"` = n'importe quel caractère chinois (avec `-u` pour `pcregrep` et `-P` pour `grep`);
- Attention: rappelez-vous que “ 。 ” ≠ “.” (pareil pour le reste de la ponctuation);