

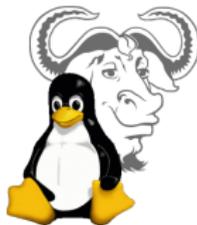
# Administration d'un système GNU / Linux

## 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

# Q grep

- grep est l'un des programmes les plus utiles de GNU;

 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");

 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);

# Q 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);

Y 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: premier exemple

### Syntaxe de grep

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

Par exemple:

- `grep bonjour fichier.txt;`

## 🔍 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: 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;`

## 🔍 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;`
- ...

## 🔍 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: 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: 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: 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: 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 ...
```

## 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;

## Variantes

Il existe quelques variantes utiles de grep:

- `pgrep`: recherche un motif parmi les noms des processus;
- `agrep`: recherche de motifs avec des “erreurs”;

## 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
```

## 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;

## 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;

## 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!

## 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 ...

# 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?

# 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 **e**ditor);

## 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 **e**ditor);

👉 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 *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;

## 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:

## 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;

## 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.

## 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!

## 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;

## 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 ...;`

## 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;

## 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:

## 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 ≠ sortie;`

## 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 ≠ sortie;`
  - ② ou `sed -i ... entree` si vous voulez écrire directement les changements dans entree;

## 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 ≠ 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;

# Expressions régulières

- Une **expression régulière** est une chaîne qui encode un *modèle* de texte;
- Exemples:

## Expressions régulières

- Une **expression régulière** est une chaîne qui encode un *modèle* de texte;
- Exemples:
  - les mots en minuscules;

# Expressions régulières

- Une **expression régulière** est une chaîne qui encode un *modèle* de texte;
- Exemples:
  - les mots en minuscules;
  - LES MOTS EN MAJUSCULES;

# 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;

# 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;

# 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;
  - ...

## 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):

Expression ↔ JavaScript ▾ 🔍 Flags ▾

// #

Text Tests **NEW** WARNING

中國歷史如果從中國的文字史首次成体系的甲骨文出现的商朝中期（前1300年算起）算起約有3,300年；从考古学上具有城市定位代表性的二里头文化遗址（前1920年）算起約有3,700年；从西周文献中的夏朝（前2070年算起）算起約有4,100年；从西周文献中傳說中的尧（前2350年算起）算起約有4400年；從孔子所言三皇五帝的傳說時代算起約有4,700年（前2698年算起）或者5300年（前3300年算起）；从已发现的最早城市良渚文化遗址算起約有5,300年[1]（也有意见认为良渚文化仍属史前时期）；從盤古、女媧、有巢氏等不確定的神話時代算起約有「五千年」（這也是傳統民間認知上的長度）；从第一座城市高庙文化彭头山遗址算起約有6800年历史；从最早的文字雏形贾湖刻符号开始算起約有8600年历史；從新石器時代的磁山文化算起約有1万年；從舊石器時代的北京猿人和藍田猿人時期算起約有68-100多萬年的历史。[2]

中国的傳說有伏羲做八卦，而近代在湖广高庙文化遗址則出土了形状为八角星的占卜盘；黃帝時代會韻造文字，而近代考古出土則發現3,300年前（前1300年）的甲骨文、4,500年前的陶文、約5,000年前至8,000年前具有文字性質的龜骨契刻符號；黃帝時代杜康造酒，而近代考古學家則在10,000年前至7,000年前的裴李崗文化賈湖遗址发现了世界上最早的釀造酒。

從政治和社會形態區分中國歷史，據考古資料顯示，約在距今六千年前的裴李崗文化晚期或者仰韶文化早期時代，中原地區從母系氏族社會漸漸過渡到父系氏族社會。同時，原始社會平等被打破，而據有文字記載的歷史，夏朝已經開始君王世襲，周朝建立完備的禮制，至東周逐漸解構，秦朝統一各國政治和許多民間分歧的文字和度量制度，並建立中央集權的專制君權統治。自漢朝起則以官文治國家直至清朝。清末以降，從西方世界東傳的科學主義、民主主義、資本主義、共產主義、社會主義等之各種新思潮始規模性流傳。20世紀初，人民興起革命終推翻數千年來的中國帝制及封建社會等傳統，並於1912年初建立首次共和國——「中華民國」。1949年10月1日，中國共產黨在大陸建立“中华人民共和国”，而中國國民黨執政的中華民國政府因國共內戰戰敗而遷至臺灣，74年以來維持兩岸分治及戰後和平格局至今。

# Applications

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

```
Expression ↔ JavaScript 🚩 Flags  
  
/前\d(4)/g  
  
Text Tests new 7 matches (1.0ms)
```

中國歷史如果從中國的文字首次成体系的甲骨文出现的商朝中期（前1300年算起）算起約有3,300年；从考古学上具有城市定位代表性的二里头文化遗址（前1920年）算起約有3,700年；从西周文献中的夏朝（前2070年算起）算起約有4,100年；从西周文献中傳說中的尧（前2350年算起）算起約有4400年；从孔子所言三皇五帝的傳說時代算起約有4,700年（前2698年算起）或者5300年（前3300年算起）；从已发现的最早城市良渚文化遗址算起約有5,300年[1]（也有意见认为良渚文化仍属史前时期）；從盤古、女娲、有巢氏等不確定的神話時代算起約有「五千年」（這也是傳統民間認上的長度）；从第一座城市高庙文化彭头山遗址算起約有6800年历史；从最早的文字雏形贾湖刻划符号开始算起約有8600年历史；從新石器時代的磁山文化算起約有1万年；從舊石器時代的北京猿人和藍田猿人時期算起約有68-100多萬年的历史。[2]

中國的傳說有伏羲做八卦，而近代在湘广高庙文化遗址則出土了形状为八角星的占卜盘；黃帝時代會創造文字，而近代考古出土則發現3,300年前（前1300年）的甲骨文、4,500年前的陶文、約5,000年前至8,000年前具有文字性質的龜骨契刻符號；黃帝時代杜康造酒，而近代考古學家則在10,000年前至7,000年前前的裴李崗文化賈湖遗址发现了世界上最早的釀造酒。

從政治和社會形態區分中國歷史，據考古資料顯示，約在距今六千年前的裴李崗文化晚期或者仰韶文化早期時代，中原地區從母系氏族社會漸漸過渡到父系氏族社會。同時，原始社會平等被打破，而據有文字記載的歷史，夏朝已經開始君王世襲，周朝建立完備的體制，至東周逐漸解構，秦朝統一各國政治和許多民間分歧的文字和度量制度，並建立中央集權的專制君權統治。自漢朝起則以文官主治國家直至清朝。清末以降，從西方世界東傳的科學主義、民主主義、資本主義、共產主義、社會主義等之各種新思潮始規模性流傳。20世紀初，人民國起革命終推翻數千年來的中國帝制及封建社會等傳統，並於1912年初建立首次共和制——「中華民國」。1949年10月1日，中國共產黨在大陸建立“中華人民共和國”，而中國國民黨執政的中華民國政府因國共內戰戰敗而遷至臺灣，74年以來維持兩岸分治及戰後和平格局至今。

On trouve 前1300年 ... mais aussi 前1920年, 前2070年, 前2350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);

## 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  $YYYY \in \llbracket 0, 9999 \rrbracket$ , donc quatre entiers dans  $[0, 9]$ ;

## 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  $YYYY \in \llbracket 0, 9999 \rrbracket$ , donc quatre entiers dans  $[0, 9]$ ;
  - le mois est  $MM \in \llbracket 1, 12 \rrbracket$ , donc un entier dans  $[0, 1]$  suivi d'un entier dans  $[0, 9]$ ;

## 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  $YYYY \in \llbracket 0, 9999 \rrbracket$ , donc quatre entiers dans  $[0, 9]$ ;
  - le mois est  $MM \in \llbracket 1, 12 \rrbracket$ , donc un entier dans  $[0, 1]$  suivi d'un entier dans  $[0, 9]$ ;
  - le jour est  $DD \in \llbracket 1, 31 \rrbracket$ , donc un entier dans  $[0, 3]$  suivi d'un entier dans  $[0, 9]$ ;

## 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  $\in \llbracket 0, 9999 \rrbracket$ , donc quatre entiers dans  $[0, 9]$ ;
  - le mois est MM  $\in \llbracket 1, 12 \rrbracket$ , donc un entier dans  $[0, 1]$  suivi d'un entier dans  $[0, 9]$ ;
  - le jour est DD  $\in \llbracket 1, 31 \rrbracket$ , 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)

## 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  $\in$   $\llbracket 0, 9999 \rrbracket$ , donc quatre entiers dans  $[0, 9]$ ;
  - le mois est MM  $\in$   $\llbracket 1, 12 \rrbracket$ , donc un entier dans  $[0, 1]$  suivi d'un entier dans  $[0, 9]$ ;
  - le jour est DD  $\in$   $\llbracket 1, 31 \rrbracket$ , 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:

## Remarques sur les dates

- Notre expression ne distingue pas les dates justes des dates fausses:
  - elle accepte des mois faux: 00, ou des nombres  $> 12$ ;

## 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  $> 31$ ;

## 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:

## 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;

## 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;

## 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;

## 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];

## 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);

## 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);

## 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, ...;

# 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`, ...;

# 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";

## 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]`, ...;

## 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	
<code>[ab]</code>	a ou b	<code>[^ab]</code>	tout sauf a et b
<code>[0-9]</code> ou <code>\d</code>	les chiffres	<code>\D</code>	tout sauf un chiffre
<code>[a-z]</code>	les lettres minuscules		
<code>[A-Z]</code>	les lettres majuscules		
<code>\w</code>	les lettres et les chiffres	<code>\W</code>	tout sauf une lettre ou un chiffre
<code>\s</code>	les espaces	<code>\S</code>	tout sauf un espace
<code>.</code>	tout sauf <code>\n</code>		

## Les répétitions

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

# Les répétitions

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

- ?: 0 ou 1 fois;

# 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;

# 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;

# 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;

# 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;

# 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;

# 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);

## 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:

## 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";

## 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]
```

## 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`

## 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`

## 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);

## 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?

## 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!

## 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";

## 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*);

## 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`;

## 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;

## 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:

## 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;

## 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` !

## 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$` !

## 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";

## 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";

## 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";

## 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;

## 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**;

## 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]`;

## 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);

## 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);