



Menu du jour :

- wooclap de recap du CM4
- encore quelques petites commandes sur les fichiers (compression, décompression, recherche)
- les droits sur les dossiers/fichiers
- les redirections sur fichiers

Wooclap de recap du CM3/4



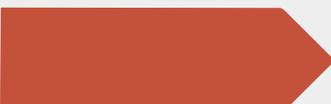
1

Allez sur wooclap.com

2

Entrez le code d'événement
dans le bandeau supérieur

Code d'événement
MSLPYC



Encore quelques commandes sur les fichiers

La commande `cmp`

`cmp` compare octet par octet les deux fichiers passés en paramètres. Cette commande renvoie « 0 » si les fichiers sont identiques, « 1 » sinon.

Exemple :

```
prompt> cmp toto toto1  
toto toto1 sont différents: octet 41, ligne 3  
prompt> echo $?  
1  
  
prompt> cmp toto toto  
prompt> echo $?  
0
```

| | toto | toto1 |
|---|---------------------------|---------------------------|
| 1 | this is the original text | this is the original text |
| 2 | line2 | line2 |
| 3 | line3 | line4 |
| 4 | line4 | happy hacking |
| 5 | happy hacking ! | GNU is not UNIX |

La commande `compress`

`compress` opère une compression visant à diminuer l'espace occupé par les différents fichiers référencés (algorithme de Lempel–Ziv–Welch). Chaque fichier *nomfichier* à compresser est remplacé par un nouveau fichier *nomfichier.Z* qui conserve les caractéristiques du fichier initial.

`compress [options] liste_fichiers`

Exemples :

```
prompt> compress -v exemple.xls
```

```
exemple.xls : -- replaced with exemple.xls.Z Compression: 24.57%
```

```
le fichier exemple.xls est compressé et remplacé par le fichier exemple.xls.Z
```

```
prompt> compress -rv abc
```

```
compresse tous les fichiers contenus dans abc et ses sous répertoires de manière récursive (-r)
```

<https://www.geeksforgeeks.org/compress-command-in-linux-with-examples/>



La commande **uncompress**

uncompress permet la **décompression** et la reconstruction d'une série de fichiers à partir de leurs formes compressées avec la commande **compress**.

uncompress *[options] liste_fichiers*

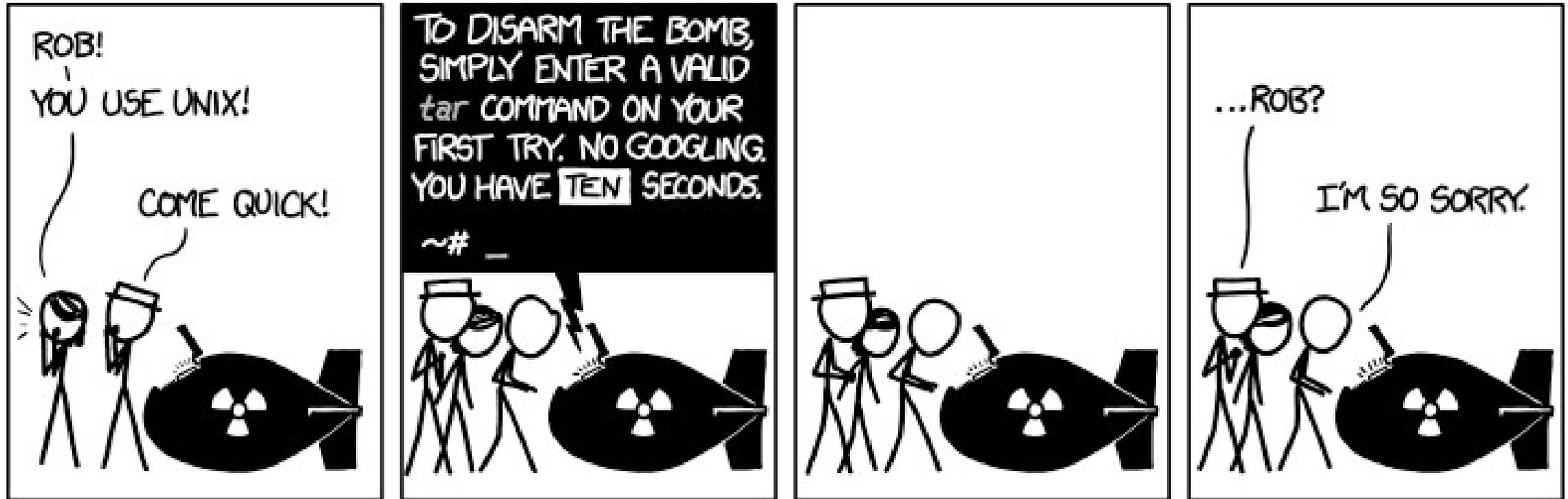
Exemple d'utilisation :

```
prompt> uncompress -v exemple.xls.Z
```

```
exemple.xls : 24.6% -- replaced with exemple.xls
```

le fichier **exemple.xls.Z** est décompressé et remplacé par le fichier **exemple.xls**

La commande tar



<https://xkcd.com/1168/>

La commande tar

tar (tape archiver) gère des archives de fichiers. La **clé** définit l'action de la commande. Elle est constituée d'un caractère définissant la **fonction** et d'autres définissant des **qualificatifs**.

```
tar [clé] liste_fichiers.tar [chemin]
```

Exemples :

```
prompt> tar cfv archive_cible.tar /etc
```

Crée une archive des fichiers du répertoire `/etc` dans le fichier `archive_cible.tar`

```
prompt> tar fxv archive_cible.tar
```

Extraction de `archive_cible.tar` dans le répertoire courant.

```
prompt> tar tvf archive_cible.tar
```

Liste les fichiers contenus dans `archive_cible.tar`

La commande **tar** – les fonctions

Il faut au moins une fonction :

-c : création d'une nouvelle archive

-r : fonction de remplacement permettant d'écrire en fin d'archive les fichiers de références données

-d: trouve les différences entre les archives

--delete : supprime de l'archive

-u : (**u**ppdate) les fichiers sont ajoutés en fin d'archive s'ils n'y figurent pas encore ou si la date de modification de la dernière version archivée est antérieure à la version du fichier sur le disque

-x: fonction d'**e**xtraction de l'archive. Si la référence examinée est une référence de répertoire, son contenu est extrait de manière récursive. Si aucune référence de fichier n'est donnée, tous les fichiers de l'archive sont extraits

... et les autres → **man tar**

La commande **tar** – les qualificatifs

- f** : l'argument suivant est interprété comme une référence de **f**ichier correspondant au nom de l'archive
- h** : les liens symboliques sont suivis (par défaut, ils ne le sont pas)
- z** : compresse avec **gzip**
- Z** : compresse avec **compress**
- v** : option « **v**erbeuse »

... et les autres → **man tar**



La commande `find`

`find` parcourt récursivement l'arborescence en sélectionnant des fichiers selon des **critères de recherche**, et **exécute des actions** sur chaque fichier sélectionné.

`find répertoire_de_départ [critère_de_recherche] action_à_exécuter`

Exemple :

`$ find ~ -print`

parcours toute l'arborescence à partir du home (~), sélectionne tous les fichiers (puisque'il n'y a aucun critère de recherche), et affiche le nom de chaque fichier trouvé.

La commande `find` – les actions possibles

- `-print` affiche le nom des fichiers sélectionnés sur la sortie standard.

Exemple : afficher toute l'arborescence de `c1`.

```
$ find /usr/c1 -print
```

- `-exec commande \;` exécute `commande` sur tous les fichiers sélectionnés. Dans la commande shell, « `{}` » sera remplacé par le nom du fichier sélectionné.

Exemple : rechercher tous les fichiers se terminant par l'extension `.o` dans l'arborescence `/usr/c1` et les détruit, puis recherche les fichiers se terminant par l'extension `.o` pour vérifier

```
$ find /usr/c1 -name '*.o' -exec rm {} \;
```

```
$ find /usr/c1 -name '*.o' -print
```

Exemple : rechercher dans les répertoires `/dev` et `/home`, tous les fichiers appartenant à `nanis`, et afficher des infos en format long.

```
$ find /dev /home -user nanis -exec ls -l {} \;
```

La commande **find** – critère de recherche

-name *modèle* sélectionne uniquement les fichiers dont le nom correspond au modèle donné.

Attention ! Le modèle doit être interprété par la commande **find** et non par le shell, donc s'il contient des caractères spéciaux pour le shell (par exemple *****), ceux-ci doivent être **échappés**.

Mauvais exemple : `$ find /usr/c1 -name *.c -print`

Le shell remplace ***.c** par la liste des fichiers finissant par **.c** du répertoire **/usr/c1**, puis va chercher dans l'arborescence donnée ces noms de fichiers.

Cela reviendra à : `$ find /usr/c1 -name f1.c f2.c f3.c -print`

Problème : **-name** n'accepte qu'un seul argument !

Par contre dans : `$ find /usr/c1 -name '*.c -print`

C'est bien ***.c** qui sera passé en argument de l'option **-name** de la commande **find**. La recherche se fera donc bien sur les trois fichiers **f1.c**, **f2.c** et **f3.c**.



La commande **find** – critère de recherche

-perm *nombre_octal* sélectionne les fichiers dont les droits d'accès sont ceux indiqués.

Exemple : Afficher tous les fichiers qui sont autorisés en lecture, écriture et exécution pour l'utilisateur propriétaire, les personnes du groupe propriétaire et tous les autres.

```
$ find /usr/c1 -perm 777 -print
```



La commande **find** – critère de recherche

-type *caractère* sélectionne les fichiers dont le type est celui indiqué.

C'est-à-dire :

- **f** pour un **fichier normal**
- **l** pour un **lien symbolique**
- **d** pour un **répertoire**
- **c** pour un fichier spécial en mode caractère
- **b** pour un fichier spécial en mode bloc

Exemple : afficher tous les répertoires et sous-répertoires de **/usr/c1**.

```
$ find /usr/c1 -type d -print
```



La commande **find** – critère de recherche

-links *nombre_décimal* sélectionne les fichiers qui ont le nombre donné de liens. Si le nombre est précédé d'un + (d'un -) cela signifie supérieur (inférieur) à ce nombre.

Exemple : pour afficher tous les fichiers qui ont plus de deux liens

```
$ find /usr/c1 -links +2 -print
```

-user *n[ou]m_utilisateur* sélectionne les fichiers dont l'utilisateur propriétaire est *nom_utilisateur* ou dont le numéro d'utilisateur (UID) est *num_utilisateur*.

Exemple : pour afficher tous les fichiers spéciaux appartenant à l'utilisateur **c1**

```
$ find /dev -user c1 -print
```



La commande **find** – critère de recherche

-inum *nombre_décimal* sélectionne les fichiers ayant pour numéro d'i-noeud *nombre_décimal*.

-newer *fichier* sélectionne les fichiers qui sont plus récents que celui passé en argument.

-atime *nombre_décimal* sélectionne les fichiers qui ont été accédés dans les *nombre_décimal* derniers jours.

-mtime *nombre_décimal* sélectionne les fichiers qui ont été modifiés dans les *nombre_décimal* derniers jours.

-size *nombre_décimal*[*c*] sélectionne les fichiers dont la taille est de *nombre_décimal* blocs. Si on post-fixe le *nombre_décimal* par le caractère *c*, alors la taille sera donnée en nombre de caractères.

La commande `find` – critère de recherche

Plusieurs critères peuvent être groupés (combinés) par les opérateurs (et).

Attention : pour le shell, ce sont des caractères spéciaux, ils doivent être échappés.

- Le **ET** logique est implicite : on met plusieurs critères à la suite et `find` sélectionne les fichiers qui répondent à tous les critères.

Exemple : afficher les fichiers se terminant par `.c` **ET** modifiés dans les 3 derniers jours.

```
$ find /usr/c1 \( -name '*.c' -mtime -3 \) -print
```

- Le **OU** logique est représenté par l'opérateur `-o`
Exemple : affiche tous les fichiers se terminant par `.txt` **OU** `.doc`.

```
$ find /usr/c1 \( -name '*.txt' -o -name '*.doc' \) -print
```

- Le **NON** logique est l'opérateur `!`
Exemple : afficher tous les fichiers **n'**appartenant **PAS** à `c1`, mais qui se trouvent dans son arborescence.

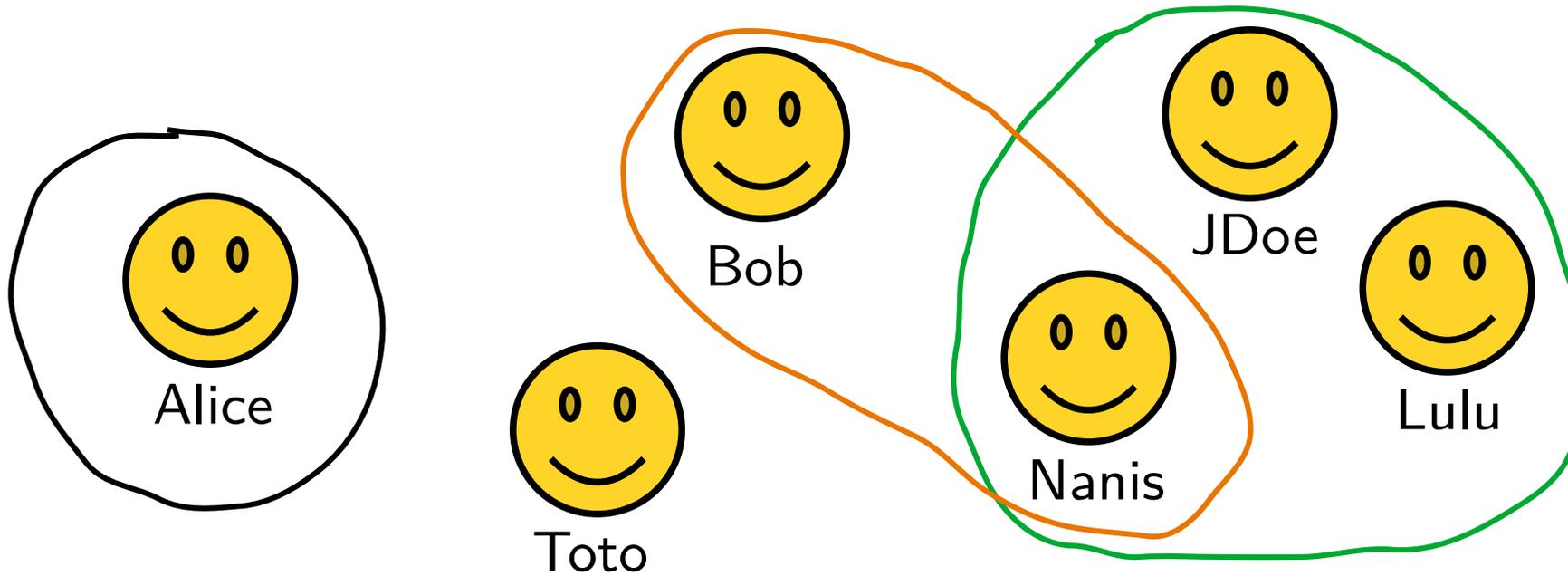
```
$ find /usr/c1 ! -user c1 -print
```



Fichiers : les droits

Rappels concernant les utilisateur.ices

- Les **utilisateur.ices** du système ont chacun.es leur **compte**.
- Leurs **droits** peuvent être administrés individuellement, ou en lot via des **groupes**.
- Il y a un compte « admin » appelé **root**, qui a **tous les droits**.
- Certain.es utilisateur.ices on le droit d'utiliser **sudo** pour prendre les droits root.





Appartenance d'un fichier / dossier

Un fichier / dossier a deux types de propriétaires :

- Un.e utilisateur.ice noté.e **u**
- Un groupe noté **g**

Les utilisateur.ices qui ne sont ni **u** ni dans **g** sont référés par « autre », noté **o** (other).

⇒ une personne donnée est soit l'utilisateur.ice **u**, soit dans le groupe **g**, soit dans **o**.

Types de modes d'accès

| Type de mode d'accès | Fichier | Dossier |
|-----------------------------|--|---|
| Lecture (r – read) | Lire le fichier (l'afficher : cat , le copier : cp) | Lister son contenu (ls) |
| Écriture (w – write) | L'éditer avec vim | Modifier les attributs du dossier et son contenu (créer un fichier, le renommer, le supprimer) /!\ ces modifs nécessitent l'accès à un inode donc faut aussi le droit x. |
| Exécution (x) | ./nomfichier | Droit de passage (cd) / Donne l'accès à l'inode d'un contenu dont on connaît le nom (ls -l fichier, stat fichier) |

cat /home/nanis/toto nécessite que la commande **cat** ouvre **toto** en mode lecture, mais aussi qu'elle ait le droit d'exécution sur « / », « **home** », et « **nanis** » pour localiser traverser chaque dossier via les inodes.

Permissions d'utiliser un mode : notations

- **Notation chaîne de caractère :**

Chaque groupe de 3 caractères correspond à une catégorie (**u**, **g**, **o**).
Si le droit pour un mode n'est pas attribué, on met un tiret.

Exemple : **rwX rwX r-x**

- **Notation binaire :**

On met 1 si le droit est donné, 0 sinon. (→ 0 s'il y a un tiret, 1 sinon).

Exemple : **111 111 101**

| lettres | binaire | octal |
|---------|---------|-------|
| --- | 000 | 0 |
| --X | 001 | 1 |
| -W- | 010 | 2 |
| -WX | 011 | 3 |
| r-- | 100 | 4 |
| r-X | 101 | 5 |
| rw- | 110 | 6 |
| rwX | 111 | 7 |

- **Notation octale :**

À partir de la notation binaire : on convertit chaque groupe de trois bits dans sa forme décimale/octale.

À partir de la chaîne de caractère : « r » vaut 4, « w » vaut 2 et « x » vaut 1.

Exemple : **775**

$$7 = 1*2^2 + 1*2^1 + 1*2^0 = 4 + 2 + 1$$

$$5 = 1*2^2 + 0*2^1 + 1*2^0 = 4 + 0 + 1$$

Wooclap



1

Allez sur wooclap.com

2

Entrez le code d'événement
dans le bandeau supérieur

Code d'événement
MSLPYC



La commande `ls -l`

`ls -l` affiche les noms de fichiers et dossier par ligne avec tout un tas d'infos, dont les droits

prompt> `cd /home` # contient un fichier « toto » et un dossier « dossier »

prompt> `ls -l toto`

```
-rw-r--r-- 1 c1 cours 342 Oct 18 15:28 toto
```

prompt> `ls -l dossier`

```
total 4
```

```
-rw-r--r-- 1 vaginay241 utilisateurs_du_domaine 18 sept. 22 18:45 contenu1-dans-dossier
```

prompt> `ls -ld dossier`

```
drwxr-xr-x 2 vaginay241 utilisateurs_du_domaine 20 sept. 22 18:47 dossier
```

prompt > `ls -l | grep dossier`

```
drwxr-xr-x 2 vaginay241 utilisateurs_du_domaine 20 sept. 22 18:47 dossier
```

Les permissions : attention !

```
[rw] .  
└─ [rwx] f_tous
```

```
$ cat f_tous
```

```
cat: f_tous: Permission non accordée
```

→ on ne peut pas lire un fichier si on n'a pas le droit « x » sur le dossier qui le contient.

```
[rwx] .  
└─ [ ] f_aucun
```

```
$ rm f_aucun
```

→ supprimer un fichier ne nécessite pas d'avoir les droit dessus.

```
[rwx] .  
└─ [x] toto.exe
```

```
$ ./toto.exe
```

```
bonjour
```

→ pas besoin du droit « r » sur le binaire pour l'exécuter.

```
[rwx] .  
└─ [x] script.sh
```

```
$ ./script.sh
```

```
/bin/bash: ./script.sh: Permission non accordée
```

→ le shell a besoin de lire un script bash pour pouvoir l'exécuter

La commande `chmod`

chmod [**change mode**] sert à modifier les droits d'accès sur un fichier ou un répertoire

Syntaxe : **chmod** *code* *chemin* où *code* est soit :

- un code octal (slide précédente)
- un code symbolique [**ugoa**][**-+=**][*modes*], relatif aux droits actuels.

Exemple d'utilisation :

```
prompt> ls -l toto
-rw-r--r-- 1 nanis cours 342 sept. 24 15:28 toto
prompt> chmod 770 toto; ls -l toto
-rwxrwx--- 1 nanis cours 342 sept. 24 15:28 toto
prompt> chmod gu-x toto; ls -l toto
-rw-rw---- 1 nanis cours 342 sept. 24 15:28 toto
```

| Personne concernée | |
|--------------------|---|
| propriétaire | u |
| groupe | g |
| autres | o |
| tous | a |
| Action | |
| ajouter | + |
| enlever | - |
| initialiser | = |
| Accès autorisés en | |
| lecture | r |
| écriture | w |
| exécution/traverse | x |



Wooclap



1

Allez sur wooclap.com

2

Entrez le code d'événement
dans le bandeau supérieur

Code d'événement
MSLPYC

La commande `chmod` – Exemple

```
prompt> ls -l fichA
-rw-rw-rw- 1 c1 cours 342 Oct 18 15:28 fichA

prompt> chmod go-w fichA; ls -l fichA
-rw-r--r-- 1 c1 cours 342 Oct 18 15:28 fichA

prompt> chmod u+x fichA; ls -l fichA
-rwxr--r-- 1 c1 cours 342 Oct 18 15:28 fichA

prompt> chmod g-r fichA; ls -l fichA
-rwx---r-- 1 c1 cours 342 Oct 18 15:28 fichA

prompt> chmod ug+rw fichA; ls -l fichA
-rwxrw-r-- 1 c1 cours 342 Oct 18 15:28 fichA

prompt> chmod g-rw fichA; ls -l fichA
-rwx---r-- 1 c1 cours 342 Oct 18 15:28 fichA

prompt> chmod ug=rw fichA; ls -l fichA
-rw-rw-r-- 1 c1 cours 342 Oct 18 15:28 fichA
```

La commande `umask`

`umask` sert à gérer les droit accordés par défaut à un création des dossier et fichiers.

Syntaxe : `umask [-S] [masque]`

Le **masque**, c'est ce qui va être soustrait aux droits par défaut à la création des futurs éléments (777 pour les dossiers et 666 pour les fichiers).

Sans l'argument masque, `umask` renvoie la valeur actuelle du masque.

Exemple :

| | | | |
|-------------------|--------------------------|------------------|----------------------------|
| droits demandés : | <code>rwX rwX rwX</code> | ou encore | <code>777</code> |
| - masque : | <code>--- -w-</code> | <code>rwX</code> | ou encore <code>027</code> |
| <hr/> | | | |
| droits accordés : | <code>rwX r-X ---</code> | ou encore | <code>750</code> |



Wooclap



1

Allez sur wooclap.com

2

Entrez le code d'événement
dans le bandeau supérieur

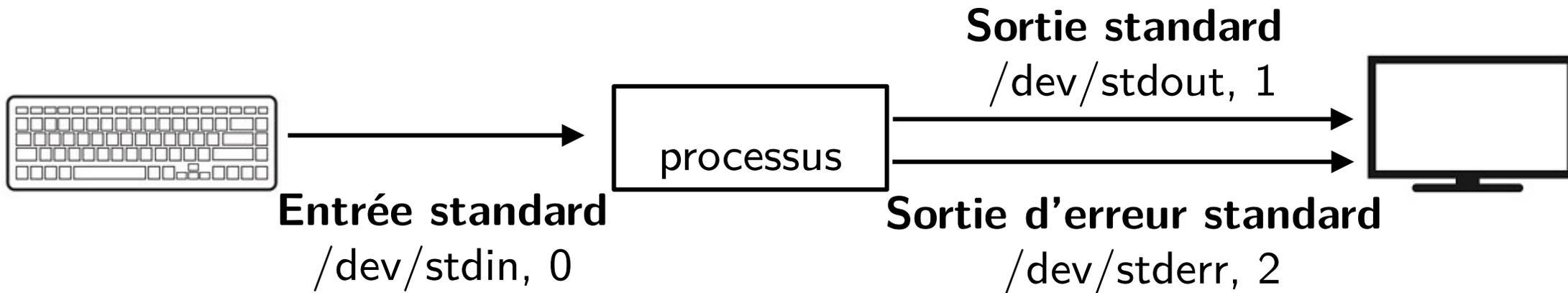
Code d'événement
MSLPYC



Utilisation des fichiers pour la redirection de commandes

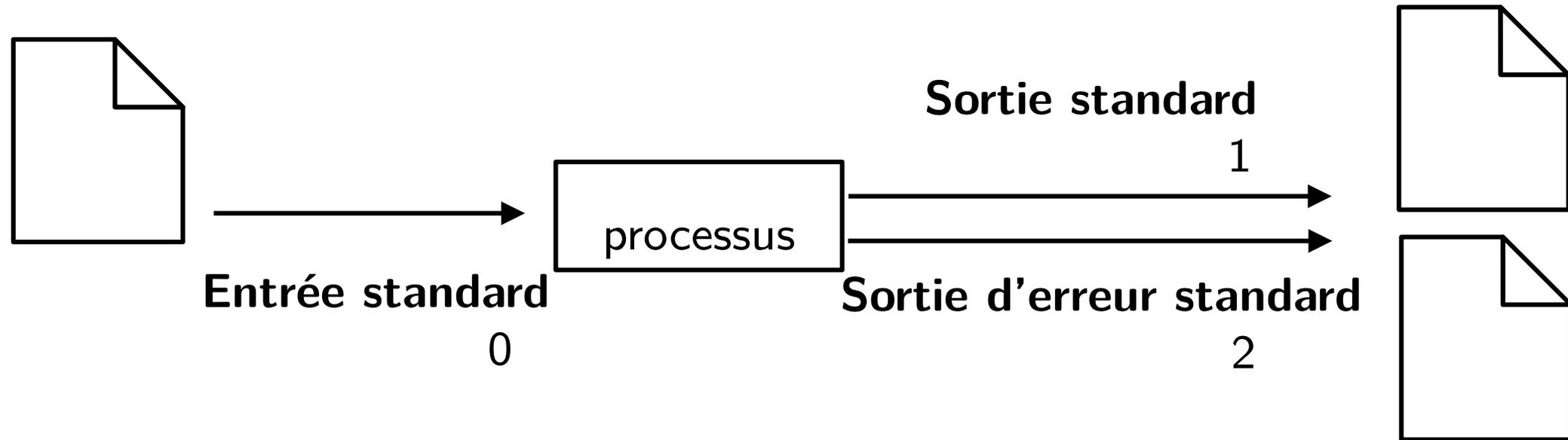
Flux de données

Tout programme qui s'exécute est par défaut associé à trois fichiers.
Chaque fichier ouvert est associé à un nombre : le **descripteur de fichier**.



Flux de données

Tout programme qui s'exécute est par défaut associé à trois fichiers.
Chaque fichier ouvert est associé à un nombre : le **descripteur de fichier**.



Redirection de la sortie standard, via >

Syntaxe : *commande* > *fichier*

Si le fichier n'existe pas, il est créé par le Shell. S'il existe déjà, le Shell détruit son contenu pour le remplacer par la sortie de la commande (« clobbering »).

Exemple :

```
prompt> who # liste les personnes connectées au système
c1 tty1 Sept 25 8:16
c2 tty3 Apr 19 2:55
c3 tty6 July 1 20:33
prompt> who > toto.txt # la commande n'affiche rien
prompt> cat toto.txt # le fichier toto.txt contient la sortie de la commande précédente
c1 tty1 Sept 25 8:16
c2 tty3 Apr 19 2:55
c3 tty6 July 1 20:33
```

Redirection de la sortie standard, via >

Pour rediriger plus d'une commande dans un fichier :

(commande1 ; commande2) > fichier

Exemple :

prompt> **(date; who) > who.txt**

→ Les sorties de **date** et **who** seront redirigées dans le même fichier : **who.txt**

prompt> **date; who > who.txt**

→ le Shell exécute **date**, affiche le résultat sur le terminal et lance ensuite **who**, en redirigeant le résultat sur le fichier **who.txt**



Redirection de la sortie standard, via `>>`

Pour ne pas écraser le contenu de fichier, mais rajouter la sortie de la commande à la fin d'un fichier (créé si besoin) : *commande* `>>` *fichier*

Exemple :

```
prompt> date > date.t ; cat date.t  
Fri Sept 27 10:45:21 MET 2023
```

```
prompt> date >> date.t ; cat date.t  
Fri Sept 27 10:45:21 MET 2023  
Fri Sept 27 10:45:23 MET 2023
```

Redirection de la sortie d'erreur standard, via 2>

Syntaxe : *commande 2> fichier*

Exemple :

```
prompt> cat fhsdofh
```

```
cat: fhsdofh: Aucun fichier ou dossier de ce nom
```

```
prompt> cat fhsdofh 2> erreur.txt % la commande n'affiche rien
```

```
prompt> cat erreur.txt
```

```
cat : fhsdofh : Aucun fichier ou dossier de ce nom
```



Redirection de l'entrée standard, via <

Syntaxe : *commande* < *fichier*

La commande va lire ses données du fichier donné en paramètre.

Exemple :

```
prompt> mail nanis < reponse
```

Mail envoyé à nanis

→ La commande **mail** lit le texte à envoyer depuis le fichier **reponse**, grâce à la redirection <, au lieu de lire les données à partir du terminal.

Redirection de l'entrée standard, via <<

Syntaxe : *commande* <<*délimiteur*

La commande va lire ses données du fichier donné en paramètre.

Exemple :

```
prompt> mail nanis <<theend  
blablablablablatheend  
theendblabla  
theend
```

Mail envoyé à nanis

→ La commande **mail** lit le texte à envoyer depuis le terminal, tant que le délimiteur n'a pas été rencontré.



Redirection : syntaxe générale

[n]redir-op word

n : descripteur de fichier (un nombre)

redir-op : un opérateur de redirection (parmi $>$, $>>$, $<$, $<<$, ...)

word : un chemin, ou un délimiteur, selon le cas.

Le standard posix :

https://pubs.opengroup.org/onlinepubs/009695399/utilities/xcu_chap02.html#tag_02_07

Redirection : droits

Soient n et m deux descripteurs de fichiers (des entiers) et *fichier* un chemin vers un fichier.

- $n < \textit>fichier$ redirige en lecture le descripteur n sur *fichier* (qui doit exister).
 $n = 0$ (l'entrée standard) par défaut.
- $n > \textit>fichier$ redirige en écriture le descripteur n sur *fichier* (qui est écrasé ou crée).
 $n = 1$ (la sortie standard) par défaut.
- $n >> \textit>fichier$ redirige en écriture le descripteur n à la fin de *fichier* (qui est crée si besoin).
 $n = 1$ (sortie standard) par défaut.

...

⇒ Il est **nécessaire d'avoir les droits adéquats** sur les fichiers utilisés en redirection !



Double redirection

Il est possible de rediriger à la fois l'entrée et la sortie :

```
prompt> wc < /etc/passwd > tmp
```

```
prompt> cat tmp
```

```
20 21 752
```

Redirection : remarques

- Une commande se contente de lire et d'écrire sur des descripteurs. Elle ne connaît pas la provenance et la destination exactes des données qu'elle lit et écrit.
- C'est l'interpréteur de commandes qui traite les demandes de redirection **avant** d'appeler la commande.
- Les redirections sont indépendantes du contexte : les caractères spéciaux « > » et « < » peuvent être situés n'importe où sur une ligne de commande.

```
prompt> who > tmp; grep 'c[12]' < tmp  
c1 tty4 Jul 31 09:46  
c2 tty2 Jul 31 09:17
```

```
prompt> > tmp who; < tmp grep 'c[12]'  
c1 tty4 Jul 31 09:46  
c2 tty2 Jul 31 09:17
```

Redirections multiples

- Un processus ne possède qu'une seule entrée, qu'une seule sortie et une seule sortie d'erreur. Donc chaque descripteur ne peut être redirigé qu'une seule fois par commande !

```
prompt> commande > fichier1 > fichier2
```

→ **fichier1** est créé mais reste vide, **fichier2** contient la sortie standard de *commande*

- La commande **tee** lit l'entrée standard et l'écrit à la fois dans la sortie standard et dans un ou plusieurs fichiers

```
prompt> commande | tee fichier1 fichier2
```

→ **fichier1** et **fichier2** sont créés et contiennent la sortie standard de *commande*

Redirection : attention aux typos

Le Shell traite les séparateurs avant les redirections ; par conséquent il y a une différence importante entre les deux commandes suivantes :

```
prompt> cmd 2> fichier
```

```
prompt> cmd 2 > fichier
```

- Dans le premier cas, on redirige la sortie d'erreur de la commande *cmd* vers *fichier*.
- Dans le second cas, on redirige la sortie standard de la commande *cmd* vers *fichier* et *2* sera considéré comme un argument de *cmd* (dû à l'espace entre le *2* et le *>*) !



TD 4 :

La politique d'accès aux
fichiers d'UNIX