

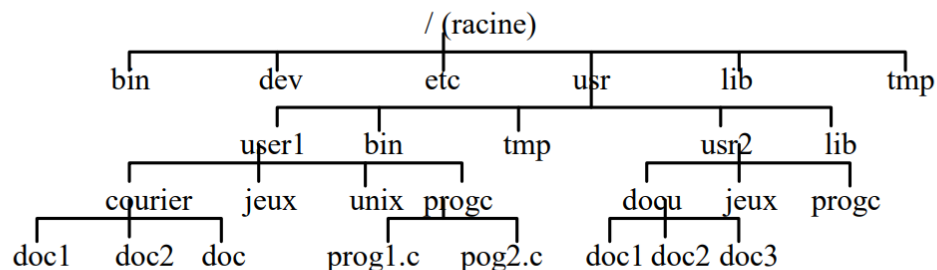
\$ cat f1 f2 f3 >>temp ajouter f1 f2 f3 dans temp
 envoie un message à des personnes provenant du fichier let
 \$ mail marie eric phil <let

\$ sort <temp trie le contenu du fichier temp

Intro. Système & Mise en Oeuvre

UNIX/Linux 58

{...} exécute les commandes dans ... par le même shell;
 (...) exécute les commandes ... par un shell fils;
 '...' prend littéralement ... ; aucun caractère spécial n'interprété ;
 "..." prend littéralement ... après interprétation de \$, '...', et \
 `...` exécute les commandes dans ... ; le résultat (sortie standard) remplace `...`



Le chemin complet du fichier **prog1.c** est : **/usr/user1/progc/prog1.**
 le chemin relatif à **doc** : **courier/doc**

L'option -l de ls imprime les permissions : \$ ls -l /etc/passwd

chmod change les permissions d'un fichier, 2 formes :

- chown, chgrp l
- Caractères génériques – ? remplace un caractère quelconque, sauf le ; – * remplace n'importe quelle chaîne, même vide, de caractères ne comportant pas – [...] remplace un caractère parmi ... , ou un caractère parmi ceux qui ne sont pas énumérés entre [...] , si le premier est un point d'exclamation.

{...} exécute les commandes dans ... par le même shell;
 (...) exécute les commandes ... par un shell fils;
 '...' prend littéralement ... ; aucun caractère spécial n'interprété ;
 "..." prend littéralement ... après interprétation de \$, '...', et \
 `...` exécute les commandes dans ... ; le résultat (sortie standard) remplace `...`

-
- **{variable:-mot}** substitue la variable si elle est définie , sinon le mot (non évalué)

```
$ t=${x:-60}          # t aura la valeur de x, sinon 60
```
 - **{variable:=mot}** idem avec de plus affectation de la variable si elle n'est pas définie

```
$ a='bonjour'
$ echo ${a:= 'au revoir'}
bonjour
$ echo ${b:= 'salut'}    # b aura la valeur 'salut'
salut
```
 - **{variable:?mot}** si la variable est initialisé et non nulle, elle sera substituée; autrement, imprime **mot** et termine le processus Shell en cours.

```
$ echo ${b:?Erreur}      # b à pour valeur initiale 'salut'
salut
$ echo ${c:?Attention!}  # c n'est pas initialisée
c : Attention!
```
 - **{variable:+mot}** si la variable est initialisé et non nulle, l'interprète substituera mot et sinon la chaîne vide, la variable n'étant pas modifiée.

```
$ echo ${b:+bonjour}
bonjour
```

Activer Windows
 Arrêtez aux paramètres n

export rend les variables visible dans les processus Shell qui seront créés par la suite (fils)

```
$ x=Bonjour
$ export x
$ sh # un shell fils
$ echo $x
Bonjour          # x sera vu dans le sous-shell
```

-
- **#** nombre de paramètres effectifs d'une commande (sous-prog.);
 - **?** code de retour d'une commande;
 - **\$** numéro de du processus Shell en cours;
 - **!** numéro du dernier processus lancé en arrière plan;
 - **HOME** nom du répertoire personnel, fournit le paramètre par défaut de **cd**;
 - **PATH** règles de recherches des commandes;
 - **TERM** type du terminal utilisé
 - **PS1** valeur de premier prompt (\$ par défaut)
 - **PS2** valeur de second prompt (> par défaut)
 - **IFS** caractères séparateur de chaînes dans le Shell.
 - Une variable peut être détruite par la commande interne **unset** (sauf **PATH**, **PS1**, **PS2**)

- **read** lit une ligne de l'entrée standard; chacun des mots est affecté successivement à une variable (dans l'ordre).

```
$ read x y z      # si on tape "une deux trois quatre cinq"
une deux trois quatre cinq
```

On aura les variables x="une", y="deux" et z="trois quatre cinq"

- **echo** affiche sa liste d'arguments sur la sortie standard; admet les caractères spéciaux (**\t \n \f \c**)

```
echo [-ne] [message ...]
```

```
echo [--help,--version]
```

-n Ne pas effectuer le saut de ligne final.

-e Interpréter les séquences de caractères précédées d'un backslash '\'

\a alerte (sonnerie)	\n saut de ligne	\c supprimer le saut de ligne final
\f saut de page	\r retour chariot	\b retour en arrière d'un caractère
\t tabulation horizontale	\v tabulation verticale	\\ backslash
\nnn le caractère de code ASCII nnn (en octal)		

- **expr** - Evalue des expressions

```
expr expression...
```

```
expr [--help,--version]
```

Incrémentation de N

```
N=`expr $N + 1`      # autres opérateurs : - * / %(mod)
```

- Opérateurs: | & < <= == != > >= + - / * % :

- **match chaîne exp_reg**

- **substr chaîne pos lg**

- **index chaîne classe_caractère**

- **length chaîne**

test renvoie une valeur 0 (vrai) ou 1 (faux) suivant l'évaluation de l'expression conditionnelle expr.

- **Commande simple:**

retourne code zéro si exécution sans erreur ou une valeur non nul à une signification propre à la commande qui l'a retournée.

- **Tube (pipe-line)** : séquence de 2 ou plusieurs commandes séparées par le "|".
- **Liste de commande**: séquence de commandes simples ou de pipe-lines séparés par l'un des caractères suivants : ; & && ||
 - **C1 ; C2** exécution séquentielle de C1 puis C2 ;
 - **C1 & C2** exécution asynchrone;
 - **C1 && C2** exécute C1; s'il n'y a pas d'erreur, exécute C2;
 - **C1 || C2** exécute C1; s'il y a une erreur, exécute C2.

- lire un choix numérique

```
echo "Votre choix (1-4) "
while echo -e "?\c"; read choix;
[ "$choix" -lt 1 -o $choix -gt 4 ]
do echo "Recommencer (1-4) "
done
```

```
case <chaîne> in
<motif> ) <liste de commandes>;
<motif> ) <liste de commandes>;
...
esac
```

- exécute la liste de commandes correspondant au premier motif satisfaisant la chaîne , puis l'exécution du **case** est terminée.

aiguillage après la lecture du choix

```
case $choix in
1) sh choix1;;
2) sh choix2;;
*) echo "Pas encore au point";;
esac
```

- **break [n]** sortir d'une boucle (tant que, pour), ou de n emboîtements de boucles;
- **continue [n]** continuer à l'itération suivante d'une boucle, ou à la nème boucle englobante;
- **exit [n]** quitter le programme avec en code retour celui de la dernière commande exécutée, ou **n** si
- **return [n]** idem pour quitter une fonction;
- **. fichier** exécute un fichier Shell dans le Shell courant;
- **:** commande vide du Shell. Son code de retour est 0
while ;; do date; done

- **eval mot** mot contient une liste de commande Shell qui seront exécutées. (sous-programme).

fichier de démarrage /etc/profile

```
MESSFIN="echo `logname` déconnecté le ; date;"
QUOTAT='TAILLE=`du -s $HOME | cut -f1`'
MESQUIT=' echo Test des quotas en cours'
eval $QUOTAS
if [ $TAILLE -gt $MAXQUOTAS ]
then
    eval $MESSFIN
exec sh
fi
```

- **set [arg1]+** fournit les valeurs des paramètres effectifs correspondant aux paramètres formels 1, 2, ...

paramétrer les sous-programmes réalisés par **eval**

```
set `date`      # range le jour de la semaine dans $1
                mois dans $2 ...
```

- **set -u [mot]** substitution d'une variable non définie (mot) provoque une erreur
- **shift [n]** les paramètres \$n+1.... (\$2, par défaut) sont renommés à partir de 1.

Redirection des E/S

- **>fichier** dirige la sortie standard vers **fichier**;
- **>> fichier** ajoute la sortie standard à **fichier**;
- **< fichier** redirige l'entrée standard vers **fichier**;
- **n> fichier** dirige la sortie du descripteur **n** vers **fichier**;
- **n>> fichier** ajoute la sortie du descripteur **n** vers **fichier**;
- **<&m** le fichier de descripteur **m** au lieu de l'entrée standard;
- **>&m** idem pour la sortie standard;
- **n> &m** fusionne la sortie du descripteur **n** à celle de **m** ;
- **n< &m** fusionne l'entrée du descripteur **n** à celle de **m** ;
- **n< fichier** dirige la sortie du descripteur **n** vers **fichier**
- **<< [-]mot** l'entrée est lue jusqu'à une ligne contenant **mot** (ou fin de fichier).- ignorer les blancs de début de ligne;

Gestion des processus et des événements

- **wait [n]** suspend l'exécution jusqu'à ce que le processus **n**, ou tous les processus lancés en arrière-plan, se terminent.
- **trap [arguments] [n]+** l'argument est une commande qui sera exécutée par le Shell lorsque le signal **n** survient;
- **trap [n]+** les trappes sont remises à leur état primitif (restaurer)
- **trap** liste des commandes associés à chaque signal.
- **exec [commande]** lance la commande par recouvrement du processus shell

```
$ exec ksh
```

```
remplacement du Shell par un processus Korn-shell
```

```
$ exec login
```

```
recommencer une session, en gardant la ligne
```

```
$ exec >session
```

```
pour ce shell la sortie standard est le fichier
```

Un texte en Shell stocké dans un fichier constitue une procédure Shell. Son activation peut se faire: `sh <nom du fichier> > <arg1>`

- **bash** a introduit des constructions familière aux programmeurs venant d'autres langages

- **[[...]]**, *test étendue*.

`[[$a -lt $b]]` est un seul élément, renvoyant un état de sortie (\$?).

- **((...))** et **let ...** renvoient 0 si les expressions arithmétiques qu'elles évaluent se résolvent en une valeur non nulle.

`let "1<2"` renvoie 0, `let "a=0"` initialise `a` à zéro et renvoie 1
`((0 && 1))` renvoie 1 (car `"0 && 1"` donne `"0"`)

- **les tableaux** à une dimension.

`aire[11]=23`

`aire[13]=37` # Les membres peuvent ne pas être consécutifs

`echo ${aire[11]}` # {accolades} nécessaires pour l'évaluation.

- **Opérateur virgule** relie une suite d'opérations arithmétiques, seul le résultat de la dernière est renvoyé.

`let "t2 = ((a = 9, 15 / 3))"` # Initialise `"a = 9"` et `"t2 = 15 / 3"`

```
wc -l <nom_du_fichier> # affiche le nombre de lignes
wc -c <nom_du_fichier> # affiche le nombre de bytes
wc -m <nom_du_fichier> # affiche le nombre de caractères
wc -L <nom_du_fichier> # indique la longueur de la plus longue ligne
wc -w <nom_du_fichier> # affiche le nombre de mots
```

2. A l'aide de la commande `sed` (qui représente un utilitaire de traitement de données capable d'utiliser les expressions régulières) effacer le contenu des lignes 4 et 7 du fichier `exemple`.

`$ sed '4d;7d' exemple`

3. Supprimer le contenu des lignes commentaires commençant par un dièse (les deux caractères slash permettent d'inclure une expression régulière).

`$ sed '/^#/ d' exemple`

4. Afficher uniquement les lignes contenant la chaîne « Contenu ».

`$ sed -n '/Contenu/p' exemple`

5. Remplacer la chaîne de caractère `#` par `*` dans le contenu du texte.

`$ sed 's/#/*/g' exemple`

Afficher la liste des fichiers se trouvant dans le répertoire `/etc` et dont le nom se termine par `.conf`, en invoquant la commande avec le paramètre `-name`.

`find /etc -name *.conf`

Retrouver la liste des fichiers spéciaux de type bloc, et de type caractère, ainsi que la liste des fichiers standards et des répertoires, se trouvant dans le répertoire `/dev`, en invoquant la commande avec le paramètre `-type`.

`find /dev -type b (bloc)`


```
find /dev -type c (char)
find /dev -type f (standard)
find /dev -type d (repertoire)
```

- Retrouver la liste de tous les fichiers dont la taille dépasse 10Mo, en invoquant la commande avec le paramètre `-size`.
`find / -size +10M (en tant que root)`
- Retrouver la liste des fichiers vous appartenant, en invoquant la commande avec le paramètre `-user`.
`find / -user ilhem`

La commande `nl` permet d'afficher un fichier texte en numérotant les lignes. Par défaut, les lignes vides ne sont pas numérotées.

```
$ tac /var/log/syslog
```

Sinon, pour inverser le contenu d'un fichier texte, il est possible d'utiliser l'enchaînement de commandes suivant :

Pour commencer, j'affiche le contenu du fichier avec la commande `cat` :

```
$ cat /var/log/syslog
```

J'enchaîne avec la commande `nl` qui permet de numéroté les lignes du fichier :

```
$ cat /var/log/syslog | nl
```

J'enchaîne avec la commande `sort` qui permet de trier le contenu du fichier numériquement (avec l'option `-n`) et en ordre inverse (avec l'option `-r`) sur la première colonne du fichier contenant les numéros de lignes générés par la commande `nl` :

```
$ cat /var/log/syslog | nl | sort -n -r
```

Il reste maintenant à supprimer la première colonne du fichier contenant les numéros de lignes avec la commande `cut` :

```
$ cat /var/log/syslog | nl | sort -n -r | cut -f2
```

Et voilà, le fichier s'affiche à l'écran dans l'ordre inverse.

`mkdir rep1 && cd rep1 && mkdir rep2` ou bien `mkdir -p rep1/rep2`

****avec la commande `ls` :**

- `-a` : afficher tous les fichiers et dossiers cachés (qui commencent par un `.`)
- `-F` : indique le type d'élément
- `-l` : liste détaillée (droits du fichier nom du prop)

-lh: taille h comme humain lisible pour un humain
 -t : trier par date de dernière modification
 -r : qui renverse l'ordre d'affichage des fichiers
 et on peut combiner tout pour avoir tout ce qu'on veut :-larth
 -mkdir : créer un dossier (mkdir nom_du_dossier) -p. Il sert à créer tous les dossiers intermédiaires. Par exemple : mkdir -p animaux/vertébres/chat
 cp -R pour copier un dossier ainsi que ses sous dossiers (majuscule !!!!!)

****grep : filtrer des données**

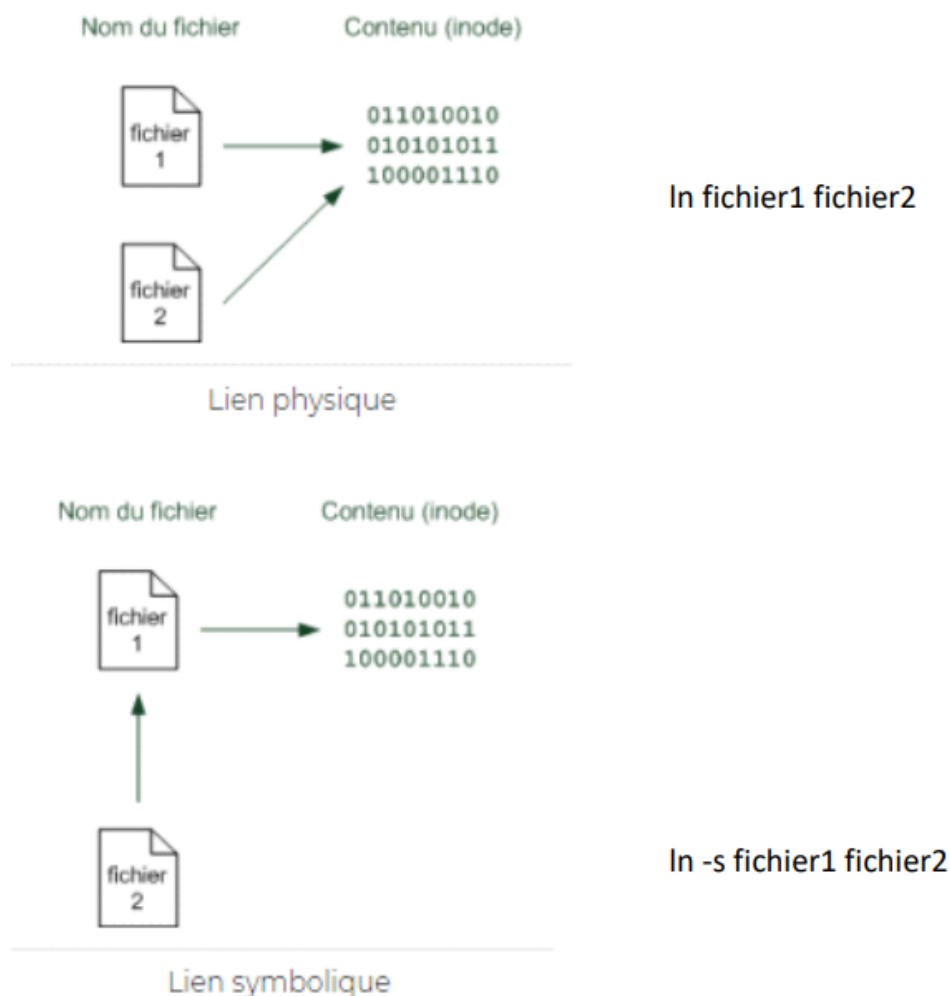
grep alias .bashrc : cette commande demande de rechercher le mot « alias » dans le fichier .bashrc et affiche toutes les lignes dans lesquelles le mot a été trouvé.

-i : ne pas tenir compte de la casse (majuscules / minuscules)

-n : connaître les numéros des lignes

-v : inverser la recherche : ignorer un mot

-r: rechercher dans tout les fichiers et les sous-dossiers.



-E pour faire comprendre à grep que l'on utilise une expression régulière
 Exemple : grep -E ^alias .bashrc (on cherche alias au début de la phrase)
 grep -E [Aa]lias .bashrc (revoie toutes les lignes qui contient alias ou Alias)

grep -E [0-4] .bashrc (renvoie toutes les lignes qui contiennent un entier entre 0 et 4)

****sort** : trier les lignes

- o : écrire le résultat dans un fichier (sort -o noms_tries.txt noms.txt)
- r : trier en ordre inverse
- R : trier aléatoirement
- n : trier des nombres

****wc** : compter le nombre de lignes

- w : nombre de mots
- c : nombre d'octets

****uniq** : supprimer les doublons uniq doublons.txt uniq doublons.txt sans_doublons.txt

- c : compter le nombre d'occurrences (uniq -c doublons.txt)

d : afficher uniquement les lignes présentes en double

****cut** : couper une partie du fichier

cut -c 2-5 noms.txt (conserver uniquement les caractères 2 à 5 de chaque ligne du fichier)

Couper selon un délimiteur :

- d : indique quel est le délimiteur dans le fichier ;
- f : indique le numéro du ou des champs à couper.

cut -d , -f 1 notes.csv

****Les variables des paramètres**

./variables.sh param1 param2 param3

- \$# : contient le nombre de paramètres ;
- \$0 : contient le nom du script exécuté (ici ./variables.sh) ;
- \$1 : contient le premier paramètre ;
- \$2 : contient de deuxième paramètre

****les tableaux**

tableau=('valeur0' 'valeur1' 'valeur2')

Pour accéder a une case du tableau : echo \${tableau[2]}

Condition	Signification
<code>\$chaine1 = \$chaine2</code>	Vérifie si les deux chaînes sont identiques. Notez que bash est sensible à la casse : « b » est donc différent de « B ». Il est aussi possible d'écrire « == » pour les habitués du langage C.
<code>\$chaine1 != \$chaine2</code>	Vérifie si les deux chaînes sont différentes.
<code>-z \$chaine</code>	Vérifie si la chaîne est vide.
<code>-n \$chaine</code>	Vérifie si la chaîne est non vide.

Pour les tests sur les chaînes de caractères, il est recommandé de mettre le nom des variables entre guillemets.

#	Condition	Signification
	<code>\$num1 - eq \$num2</code>	Vérifie si les nombres sont égaux (equal). À ne pas confondre avec le « = » qui, lui, compare deux chaînes de caractères.
	<code>\$num1 - ne \$num2</code>	Vérifie si les nombres sont différents (not equal). Encore une fois, ne confondez pas avec « != » qui est censé être utilisé sur des chaînes de caractères.
	<code>\$num1 - lt \$num2</code>	Vérifie si <code>num1</code> est inférieur (<) à <code>num2</code> (lower than).
	<code>\$num1 - le \$num2</code>	Vérifie si <code>num1</code> est inférieur ou égal (<=) à <code>num2</code> (lower or equal).
	<code>\$num1 - gt \$num2</code>	Vérifie si <code>num1</code> est supérieur (>) à <code>num2</code> (greater than).
	<code>\$num1 - ge \$num2</code>	Vérifie si <code>num1</code> est supérieur ou égal (>=) à <code>num2</code> (greater or equal).

#	Condition	Signification
	<code>-e \$nomfichier</code>	Vérifie si le fichier existe.
	<code>-d \$nomfichier</code>	Vérifie si le fichier est un répertoire. N'oubliez pas que sous Linux, tout est considéré comme un fichier, même un répertoire !
	<code>-f \$nomfichier</code>	Vérifie si le fichier est un... fichier. Un vrai fichier cette fois, pas un dossier.
	<code>-L \$nomfichier</code>	Vérifie si le fichier est un lien symbolique (raccourci).
	<code>-r \$nomfichier</code>	Vérifie si le fichier est lisible (r).
	<code>-w \$nomfichier</code>	Vérifie si le fichier est modifiable (w).
	<code>-x \$nomfichier</code>	Vérifie si le fichier est exécutable (x).
	<code>\$fichier1 - nt \$fichier2</code>	Vérifie si <code>fichier1</code> est plus récent que <code>fichier2</code> (newer than).
	<code>\$fichier1 - ot \$fichier2</code>	Vérifie si <code>fichier1</code> est plus vieux que <code>fichier2</code> (older than).

```
#for i in `seq 1 $2`
for ((i=1;i<=$2;i++))
```

- `cmd1 ; cmd2` : enchaînement séquentiel ; `cmd2` est exécutée lorsque `cmd1` est terminée
- `cmd1 & cmd2` : enchaînement parallèle ; `cmd1` et `cmd2` sont lancées en parallèle
- `cmd1 && cmd2` : et ; `cmd2` est exécutée si `cmd1` retourne vrai
- `cmd1 || cmd2` : et ; `cmd2` est exécutée si `cmd1` retourne faux

-f :est un fichier ordinaire -d :est un répertoire -p :est une représentation interne d'un dispositif de communication -c :est un pseudo-fichier du type acc`es caract`ere par caract`ere -b :est un pseudo-fichier du type acc`es par bloc -L :est un lien symbolique -u : son Set UID=1 -g :son Set GID=1 -k :son Sticky Bit=1 -S :est non-vide

Tests de chaines `test chaîne` (ou `[chaîne]`) : vraie si `chaîne` est une chaîne vide -z `chaîne` :vraie si `chaîne` est une chaîne vide -w `chaîne` :vraie si `chaîne` est une chaîne non-vide

le fichier `/etc/passwd` qui contient la liste des utilisateurs de la machine.

La commande `cut` est l'un des outils de filtrage de texte présents dans Linux et UNIX.

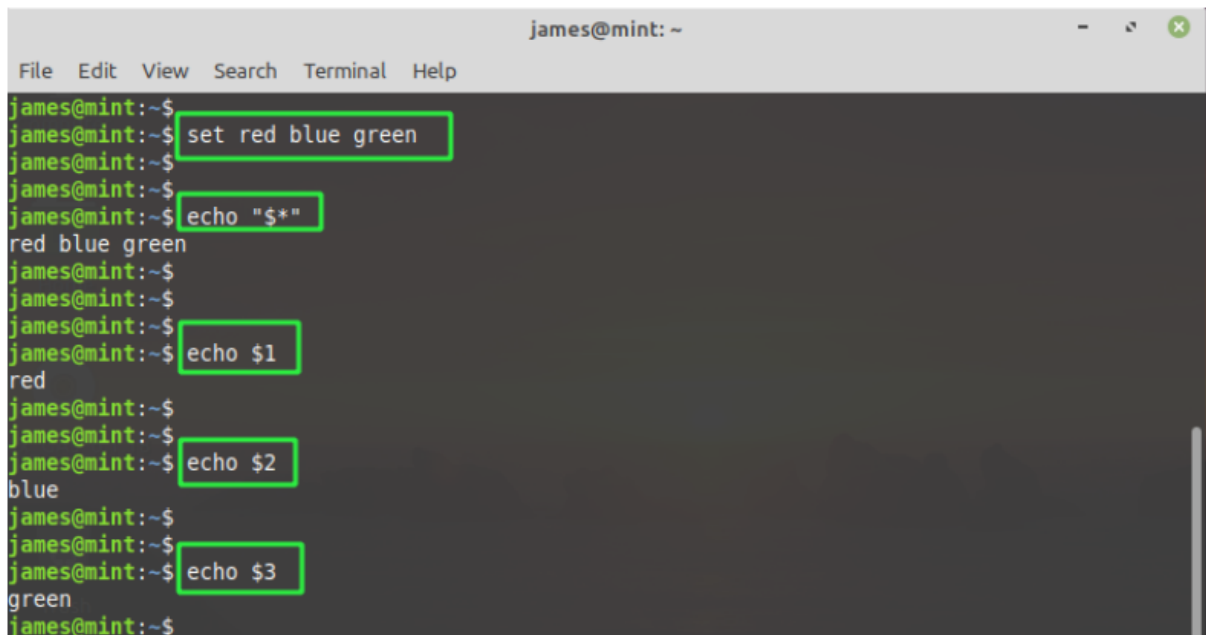
Elle s'utilise pour extraire des colonnes spécifiques des lignes de texte. Vous passez du texte à l'aide de fichiers ou la sortie d'une autre commande à la commande `cut`.

Ensuite elle imprime les données à la sortie standard du [terminal](#).

```
cut options [fichier]
```

Voici la liste des options disponible pour la commande `cut` :

Options	Description
-f (- champs)	Spécifiez les champs que vous souhaitez extraire.
-c (- caractères)	Spécifiez les caractères que vous souhaitez extraire.
-b (- octets)	Fournissez les octets que vous souhaitez extraire.
-d (-Délimiteur)	Ici, vous spécifiez le délimiteur que vous souhaitez utiliser avec la commande CUT. Par défaut, l'onglet est considéré comme un délimiteur.
-complement	Il est utilisé pour obtenir des colonnes non spécifiées par -f, -c ou -b Options.
-output-delimiter	Par défaut, la commande CUT utilise le délimiteur d'entrée en tant que délimiteur de sortie, mais vous pouvez modifier ce comportement en utilisant cette option.

A screenshot of a terminal window titled 'james@mint: ~'. The window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows a series of commands and their outputs: 'set red blue green' sets three variables; 'echo "\$*"' prints 'red blue green'; 'echo \$1' prints 'red'; 'echo \$2' prints 'blue'; and 'echo \$3' prints 'green'.

```
james@mint:~$ set red blue green
james@mint:~$ echo "$*"
red blue green
james@mint:~$
james@mint:~$
james@mint:~$ echo $1
red
james@mint:~$
james@mint:~$ echo $2
blue
james@mint:~$
james@mint:~$ echo $3
green
james@mint:~$
```

vous permet de créer, de fusionner ou d'imprimer des fichiers dans l'écran de résultat standard ou vers un autre fichier et bien plus encore.

```
cat [OPTION] [FICHIER]
créer un fichier avec cat
cat > nomdufichier.txt
```

Pour ajouter plusieurs lignes de texte, il suffit d'appuyer sur la touche **Entrée** à la fin de chaque ligne. Une fois que vous avez terminé, appuyez sur **CTRL+D** pour quitter le fichier.

visualiser le contenu d'un fichier

```
cat nomdufichier.txt
```

Pour éviter de faire défiler des fichiers de grande taille, vous pouvez ajouter l'option | **more** pour obtenir un affichage plus ou moins grand

redirection du contenu à l'aide de cat

```
cat source.txt > destination.txt
```

concaténation des fichiers

```
cat source1.txt source2.txt > destination.txt
```

option de cat

- E marquer les fins de ligne en affichant le caractère \$ à la fin de chaque ligne.

- n afficher le contenu d'un fichier avec les numéros de ligne au début de chacun d'entre eux.

- v Pour afficher tous les caractères non imprimables

-s Pour supprimer les lignes vides répétées et l'espace de sécurité sur votre écran,

-b Pour afficher les lignes non vides avec leur numéro de ligne imprimé avant,

La commande `tr` peut effectuer des opérations telles que la suppression de caractères répétés, la conversion de majuscules en minuscules et le remplacement et la suppression de caractères de base.

```
tr OPTION... SET1
```

`tr` accepte deux jeux de caractères, généralement de même longueur, et remplace les caractères des premiers jeux par les caractères correspondants du deuxième jeu.

Un `SET` est essentiellement une chaîne de caractères, y compris les caractères spéciaux avec barre oblique inverse.

```
echo 'linuxize' | tr 'lin' 'red'
```

Chaque occurrence de `l` est remplacée par `r`, `i` par `e` et `n` par `d` :

Lorsque l'option `-c` (`--complement`) est utilisée, `tr` remplace tous les caractères qui ne sont pas dans `SET1`.

```
echo 'linuxize' | tr -c 'li' 'xy'
```

```
liyyyiyyy
```

L'option `-d` (`--delete`) indique à `tr` de supprimer les caractères spécifiés dans `SET1`. Lors de la suppression de caractères sans compression, spécifiez un seul jeu.

```
echo 'Linuxize' | tr -d 'liz'
```

Le caractère `L` n'est pas supprimé car l'entrée comprend un `L` majuscule tandis que le caractère `l` dans le SET est en minuscule.

```
Lnuxe
```

L'option `-s` (`--squeeze-repeats`) remplace une séquence d'occurrences répétées par le jeu de caractères du dernier SET.

Dans l'exemple suivant, `tr` supprime les caractères d'espace répétés:

```
echo "GNU \ Linux" | tr -s ' '
```

```
GNU \ Linux
```

Lorsque SET2 est utilisé, la séquence du caractère spécifié dans SET1 est remplacée par SET2.

```
echo "GNU \ Linux" | tr -s ' ' '_'
```

```
GNU_\_Linux
```

`tr -cd` supprime tous les caractères non numériques

documents mahmoud