

les **heredoc** en bash sont une manière de passer plusieurs lignes de texte ou de code à une commande interactive, comme 'cat'.

Ils permettent d'inclure des blocs de texte volumineux dans les scripts sans avoir à se soucier de l'échappement des caractères spéciaux etc

SYNTAXE
DE BASE =

[command] << [-] 'delimiter'
here-document
delimiter) → input

delimiter = peut-être n'importe quel chaîne de caractère mais le plus souvent utilisé c'est EOF ou END. Si le delimiter est entre guillemets simples (' ') toutes les variables/commandes/caractères spéciaux seront substitués par le shell avant que les lignes du heredoc soient passées à la commande.
le delimiter marque la fin du heredoc.

[-] = pour ignorer les espaces blancs au début des lignes du heredoc. Cela permet l'indentation lors de l'écriture de heredoc dans les scripts shell, tout en évitant les espaces blancs en début de ligne.

[commande] = facultatif. Peut-être n'importe quel commande qui utilise la redirection.

exemple =
avec cat

```
cat << EOF  
ligne 1  
ligne 2  
EOF
```

→ 'cat' affiche deux lignes de texte passées via le heredoc

UTILISATION DANS
DES FONCTIONS

= les heredocs peuvent être utilisés pour passer plusieurs arguments à une fonction.

```
function details() {  
    read name  
    read age  
    read nationality  
}
```

```
details << EOF  
Harry  
23  
Brit  
EOF
```

la fonction "details" lit 3 variables à partir du heredoc.

CRÉER UN
FICHIER

= Créez un fichier "filename.sh" avec un en-tête bash.

```
<< EOF > filename.sh  
#!/bin/bash  
EOF
```

UTILISER DES
VARIABLES
D'ENVIRONNEMENTS

=

```
cat << EOF  
Current working: $PWD  
logged as: $(whoami)  
EOF
```



output = Current working: /home/
logged as: lucie

≠ Si on met le here-doc entre double quotes
les commandes ne seront pas substituées

cat << "EOF"

Current working : \$PWD

logged as : \$(whoami)

EOF

→ output =

Current working : \$PWD

logged as : \$(whoami)

EOF

INDENTER POUR
ÉCRIRE DU CODE =

if true; then
cat <<- EOF
line with a leading tab.
EOF
fi

REDIRIGER
L'OUTPUT =

cat << EOF > file.txt
WRiting in a file
EOF

⚠ Si le fichier n'existe pas il sera créée.
En utilisant 1 > on overwrite.
" 2 >> on écrit à la suite.

HEREDOC =
et PIPE =

cat << 'EOF' | sed 's/l/e/g'
Hello
girls and ladies
EOF

→ output : Heeee
eieee and eadiee

HEREDOC EXECUTION ORDER =

- 1 → Start execution : Shell starts executing the cmd that precedes `<<`.
- 2 → Begin here document : When shell encounters `<<`, it expects input. The word immediately after `<<` will be used as the delimiter to end document.
- 3 → Input collection : Shell collects all lines of input that follow until it encounters the delimiter. These lines can include commands, variables, or any text you want to pass as input.
- 4 → Command substitution : If the heredoc includes cmd substitutions `$(...)`, like `$(ls)` for ex, this are executed at this stage and their output will be included in the input to [cmd]
- 5 → Execute command : Once the shell has collected all the input lines and processed any cmd substitution it passes the entire block of input to the [cmd]
- 6 → [cmd] execution : The [cmd] receives the input from the here doc and processes it. For example cat receive the list of files from `$(ls)`
- 7 → End of heredoc : After the entire content has been passed to the cmd, the shell sees the delimiter.

the [cmd] ends processing the input from heredoc.

8 → Continue Script execution : When more commands are in script/shell , they're executed as usual after the heredoc.

⚠ If we use the backslash before the delimiter, it is ignored = /EOF ≠ EOF

PIPING with HEREDOCS

1 → Input piping : The stdout of the cmd on the left (ls) is sent as stdin to the cmd on the right (grep). Here, we'll use a pipe to send the output as input to heredoc - cat << EOF
\$(ls | grep ".txt")
EOF

2 → Output piping : It passes the output of one cmd as input to another cmd, all within the heredoc. We can also pipe the output of a heredoc into another cmd. cat << EOF | grep "grape"

apple
banana
grape
orange
EOF

→ The lines within heredoc are input stdin to cat.

→ Cat outputs the lines which are piped to grep

→ grep searches in its stdin which is the stdout of cat
→ The final output will be the stdout of grep.

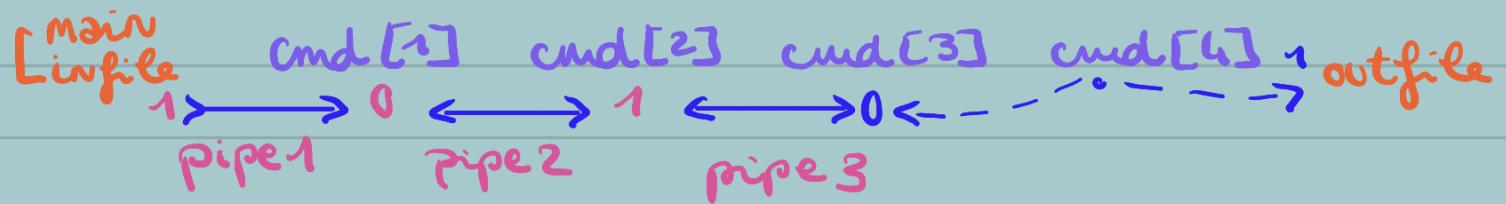
⚠ If we have an issue with heredoc → bash -x

? Dans le sujet il est écrit "gérez << et >> quand le premier paramètre est "here-doc""
↳ donc on doit juste faire ça ???

↳ On gère que le here-doc avec pipe!

du coup dans mon parsing si

PIPELINE



→ count
→ Nombre de commandes - 1 = indique nombre de boucle pour fork et sert d'index pour Rediriger vers les fonctions spécifiques.

→ On crée un pipe sauf si index == count

→ On fork

↳ Si le fork a réussi

main cmd[1] cmd[2] outfile
P1 P2 cmd 2

index 0 count2

→ créer les pipes nécessaires
pipex → boucle selon nombre d'argument
Count

→ fork

→ Si index == 0 - check-access (infile)
openfile →

→ Si index == Count - open (outfile)
└ check-access

→ ft-dup (index, count)

→ close-files

→ executing

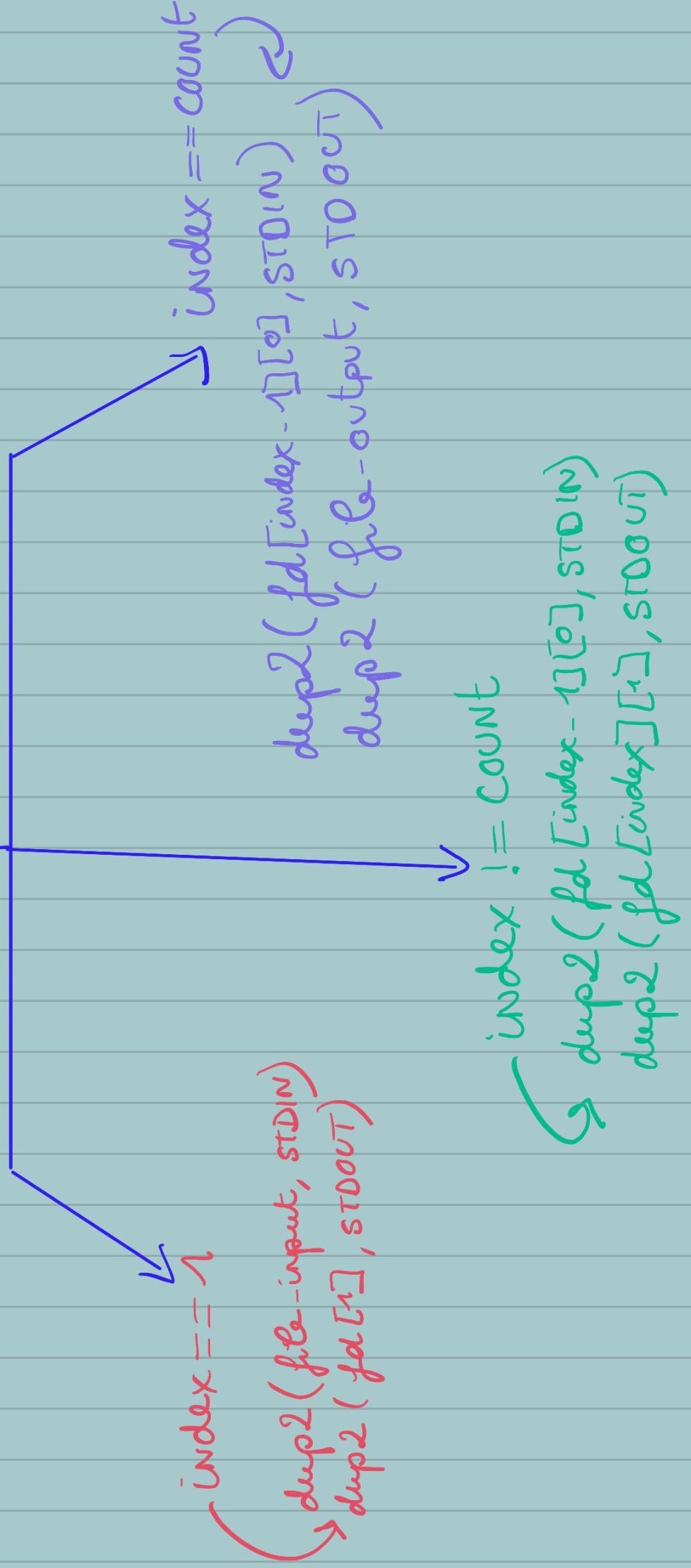
→ index ++;

→ current-node = next;

→ close-files

→ ft-wait-all (boucle pour wait
tous les child);

$fht_dup(index, count)$



Dans mon parsing deux cas :

si $\text{av}[1] == \text{here-doc}$

↳ Limiter == $\text{av}[2]$
↳ Content == gnl

si $\text{av}[1] != \text{here-doc}$

suit naturellement
soon const -

Si argc == 5 → here-doc = ERROR (non)
peut être traités de → suit son caout avec 2 cmd.
la même façon ??
si argc > 5 && → here-doc = ERROR
aw[1] != here-doc → gérer plusieurs erreurs

si argc > 5 && → parsing différent.
aw[1] == here-doc

here-doc fd[nb-arg-3][2]
≠ fd[nb-arg-2]

0 1 2 3 4 5
here-doc LIMITER cmd cmd1 file = 6
file1 cmd1 cmd2 file2 = 5

ft-parsing()

↳ here-doc == 0

↳ ft-parsing-here-doc()

↳ limiter = argv[2] // protéger si null

file-input = open here-doc

Si existe pas crée le

new = get-next-line(0, 0)

Si dans new pas de limiter

waite(file-input, new, strlen(new))

ft-parsing-cmd(pipex, argv, j, :)

i = index pour argv de sorte différent

si heredoc = 3

sinon = 2

j = i pour ajouter le bon index
aux commandes.

→
ft-parsing-fd

0 1 2 3 4
./a.out infile cmd cmd2 outfile

1 2 3 4 5 6
./a.out here-doc EOF "grep hello" cat outfile.txt
0 1 2 3 4 5

argc = 6 nb-arg = 5 nb-cmd = 2

infile = av[1];

outfile = av[];

S-1

index = 0 = cmd 1

index = 1 = cmd2

1 → open temporary file = Write ONLY

2 → write content in here doc

3 → open the file READ ONLY

4 → close the original file
+ unlink

5 → Redirect the file to st

open WONLY ③

dup(3) = 4

write(4 " ")

close(4);

open RONLY 4

close(3)

unlink(name)

dup(4, 0)

close(4)

Waitpid (pid, &status, 0)

if (WIFEXITED (status))

status = WEXITSTATUS (status)

return (status)

fd [3] = { 0, 1, % }
 | | |
 1 2 3

j=0 j=1 j=2

fd [3] { cmd1, cmd2, 0 }
 0/1 1/2 2/3

fd [3] [2]

0 | X
0 1 2

problème si env -i ./pipex infile cmd

0 1 2 3 4

du coup cmd = av[2] = pipex
outfil = av[arg-1]

pas ça le problème

limiteur = lim

problème si j'ai un input sur une seule ligne comme = yes/nlim limy/nlim
bah du coup mon programme break et n'écris pas dans le fd/outfile

du coup il faut écrire une fonction qui imprime caractère par caractère et puis break avant le lim.

Si lim dans input \rightarrow write every

Ce que je veux comparer

lim+1 "% " ? si oui = lim
si non continue

1/n

2/n

lime/n
1 2 3 4

lim/n = /n
1 2 3 4

