

# **Formation Bash Scripting - Séance 1 Matinée**

**Bienvenue dans la Formation  
Bash Scripting**

**Séance 1 - Fondamentaux et premiers pas  
Matinée (3h30)**

*Niveau BAC+2 TSSR - 35h de formation*

# Objectifs de la matinée

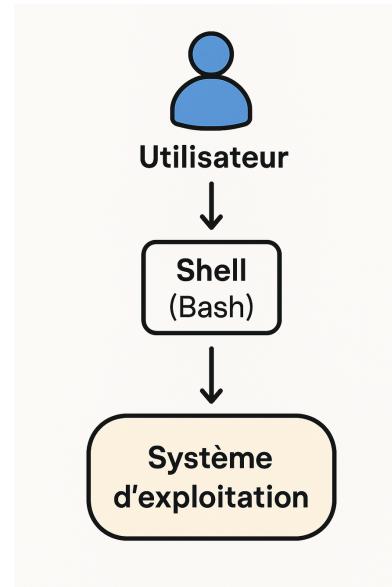
- Comprendre le rôle du shell dans les systèmes Unix/Linux
- Maîtriser la structure d'un script Bash
- Découvrir les variables et types de données
- Créer ses premiers scripts interactifs

# **Introduction au Shell et Bash**

## **(1h30)**

# Qu'est-ce qu'un Shell ?

- **Interface** entre l'utilisateur et le système d'exploitation
- **Interpréteur de commandes** en ligne de commande
- **Langage de programmation** pour automatiser les tâches



# Historique des Shells

<b>Shell</b>	<b>Année</b>	<b>Caractéristiques</b>
<b>sh</b>	1971	Premier shell Unix (Bourne Shell)
<b>csh</b>	1978	Syntaxe similaire au C
<b>bash</b>	1989	Bourne Again Shell
<b>zsh</b>	1990	Shell moderne et extensible

# Pourquoi Bash ?

- ✓ **Standard** sur la plupart des distributions Linux
- ✓ **Compatibilité** avec les scripts sh
- ✓ **Fonctionnalités avancées** (tableaux, fonctions)
- ✓ **Largement documenté** et supporté

# Shell Interactif vs Scripts

## Shell Interactif

```
● ● ●  
1 $ ls -la  
2 $ cd /home/user  
3 $ grep "error" /var/log/syslog
```

## Script Bash

```
● ● ●  
1 #!/bin/bash  
2 ls -la  
3 cd /home/user  
4 grep "error" /var/log/syslog
```

# Variables d'Environnement Essentielles



```
1 echo $PATH          # Chemins des exécutables  
2 echo $HOME          # Répertoire personnel  
3 echo $USER          # Nom d'utilisateur  
4 echo $PWD           # Répertoire courant  
5 echo $SHELL          # Shell utilisé
```

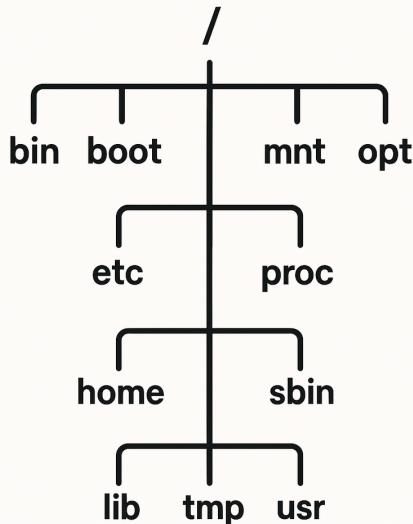
**Exercice pratique :** Affichez toutes ces variables sur votre système

# Navigation et Commandes de Base



```
1 pwd          # Afficher le répertoire courant
2 ls -la       # Lister les fichiers détaillés
3 cd /path/to/dir # Changer de répertoire
4 mkdir mon_dossier # Créer un répertoire
5 touch fichier.txt # Créer un fichier vide
```

Arborescence Linux typique



# **Structure d'un Script Bash**

## **(2h)**

# Anatomie d'un Script Bash



```
1 #!/bin/bash
2 # Ceci est un commentaire
3 # Script : mon_premier_script.sh
4 # Auteur : [Votre nom]
5 # Date : $(date)
6
7 echo "Hello World!"
8 echo "Mon premier script fonctionne !"
```

# Le Shebang (#!)



```
1 #!/bin/bash          # Standard
2 #!/usr/bin/env bash # Portable
3 #!/bin/sh            # Compatible POSIX
```

**Rôle :** Indique au système quel interpréteur utiliser

**Emplacement :** Toujours en première ligne du script

# Permissions d'Exécution



```
1 # Créer le script
2 nano mon_script.sh
3
4 # Rendre exécutable
5 chmod +x mon_script.sh
6
7 # Vérifier les permissions
8 ls -l mon_script.sh
9 -rwxr-xr-x 1 user user 156 Jan 15 10:30 mon_script.sh
```

# Méthodes d'Exécution



```
1 # Méthode 1 : Exécution directe
2 ./mon_script.sh
3
4 # Méthode 2 : Via bash
5 bash mon_script.sh
6
7 # Méthode 3 : Chemin absolu
8 /home/user/scripts/mon_script.sh
```

# Les Commentaires

```
1 #!/bin/bash
2
3 # Commentaire sur une ligne
4 echo "Hello" # Commentaire en fin de ligne
5
6 # Bloc de commentaires
7 # Ceci est un commentaire
8 # sur plusieurs lignes
9 # pour expliquer le code
10
11 echo "World"
```

**Bonnes pratiques :** Commentez le "pourquoi", pas le "quoi"

# Bonnes Pratiques de Documentation



```
1 #!/bin/bash
2 #
3 # Script de sauvegarde automatique
4 #
5 # Description : Effectue une sauvegarde des fichiers utilisateur
6 # Auteur      : John Doe
7 # Version     : 1.0
8 # Date        : 2025-01-15
9 # Usage       : ./backup.sh [source] [destination]
10 #
11
12 # Configuration
13 SOURCE_DIR="/home/user/documents"
14 BACKUP_DIR="/backup"
```



# TP Pratique

Premier script "Hello World"

Durée : 30 minutes

# **Exercice 1.1 : Script de Présentation**

**Objectif :** Créer un script qui affiche des informations personnalisées

**À réaliser :**

- Créez `presentation.sh`
- Affichez votre nom
- Affichez la date/heure actuelles
- Affichez le nom de la machine
- Affichez le répertoire de travail

# Solution Exercice 1.1



```
1 #!/bin/bash
2 # presentation.sh - Script de présentation personnalisé
3
4 echo "Bonjour, je suis [Votre Nom]"
5 echo "Date et heure : $(date)"
6 echo "Nom de la machine : $(hostname)"
7 echo "Répertoire actuel : $(pwd)"
```



```
1 chmod +x presentation.sh
2 ./presentation.sh
```

# Exercice 1.2 : Information Système

**Objectif :** Découvrir les commandes système de base

**À réaliser :**

- Créez `info_systeme.sh`
- Affichez l'utilisateur connecté
- Affichez l'espace disque
- Affichez la charge système
- Affichez les dernières connexions

# Solution Exercice 1.2

```
1 #!/bin/bash
2 # info_système.sh - Informations système
3
4 echo "==== INFORMATIONS SYSTÈME ==="
5 echo "Utilisateur connecté : $(whoami)"
6 echo ""
7 echo "==== ESPACE DISQUE ==="
8 df -h
9 echo ""
10 echo "==== CHARGE SYSTÈME ==="
11 uptime
12 echo ""
13 echo "==== DERNIÈRES CONNEXIONS ==="
14 last -5
```

# **Variables et Types de Données**

## **(2h)**

# Déclaration de Variables



```
1 # Déclaration simple
2 nom="John"
3 age=25
4 ville="Paris"
5
6 # Pas d'espaces autour du =
7 # ✓ Correct
8 variable="valeur"
9
10 # ✗ Incorrect
11 variable = "valeur"
```

# Utilisation des Variables



```
1 nom="Alice"
2 age=30
3
4 # Affichage simple
5 echo $nom
6 echo $age
7
8 # Affichage avec accolades (recommandé)
9 echo ${nom}
10 echo "Bonjour ${nom}, vous avez ${age} ans"
```

# Variables Locales vs Globales

```
1 #!/bin/bash
2
3 # Variable globale
4 nom_global="Global"
5
6 ma_fonction() {
7     # Variable locale
8     local nom_local="Local"
9     echo "Dans la fonction : $nom_local"
10    echo "Accès global : $nom_global"
11 }
12
13 ma_fonction
14 echo "Hors fonction : $nom_local" # Vide !
```

# Chaînes de Caractères

```
● ● ●  
1 # Guillemets simples : littéral  
2 echo 'Hello $USER'      # Affiche : Hello $USER  
3  
4 # Guillemets doubles : expansion  
5 echo "Hello $USER"       # Affiche : Hello john  
6  
7 # Sans guillemets : attention aux espaces  
8 fichier=mon fichier.txt # ✗ Erreur  
9 fichier="mon fichier.txt" # ✓ Correct
```

# Échappement de Caractères

**Pourquoi échapper ?** Certains caractères ont une signification spéciale en Bash



```
1 # Caractères réservés qui posent problème :
2 echo $USER          # $ = expansion de variable
3 echo "test"         # " = délimiteur de chaîne
4 echo 'test'         # ' = chaîne littérale
5 echo test*          # * = wildcard (tous les fichiers)
6 echo test|wc        # | = pipe
7 echo test&          # & = arrière-plan
8
9 # Solutions d'échappement :
10 echo "Prix : 10\$ seulement"           # \ échappe le $
11 echo "Guillemets \"doubles\"" inclus" # \" pour " littéral
12 echo 'C'\''est important'            # Combiner ' et \
13 echo "Fichier\*.txt"                 # \* pour * littéral
14
15 # Séquences d'échappement avec echo -e :
16 echo -e "Ligne 1\nLigne 2"           # \n = nouvelle ligne
17 echo -e "Colonne1\tColonne2"         # \t = tabulation
18 echo -e "Son\bip"                   # \a = bip sonore
```

**Règle :** Quand un caractère a un sens spécial, utilisez \ pour le rendre littéral

# Variables Numériques



```
1 # Déclaration
2 nombre1=10
3 nombre2=5
4
5 # Opérations arithmétiques
6 resultat=$((nombre1 + nombre2))
7 echo "Somme : $resultat"
8
9 # Autres opérations
10 echo "Différence : $((nombre1 - nombre2))"
11 echo "Produit : $((nombre1 * nombre2))"
12 echo "Division : $((nombre1 / nombre2))"
13 echo "Modulo : $((nombre1 % nombre2))"
```

# Opérations Arithmétiques Avancées



```
1 # Méthode 1 : $(( ))
2 calcul=$((10 * 3 + 5))
3
4 # Méthode 2 : let
5 let "resultat = 10 * 3 + 5"
6
7 # Méthode 3 : expr (ancienne)
8 resultat=$(expr 10 \* 3)
9
10 # Incrémentation
11 compteur=0
12 ((compteur++))      # compteur devient 1
13 ((compteur+=5))     # compteur devient 6
```



# TP Pratique

## Calcul Simple et Manipulation de Chaînes

**Durée :** 45 minutes

# **Exercice 1.3 : Calculatrice Simple**

**Objectif :** Manipuler les variables numériques

**À créer :** calculatrice.sh

**Fonctionnalités :**

- Définir deux nombres
- Calculer somme, différence, produit, modulo
- Format d'affichage : "5 + 3 = 8"

# Solution Exercice 1.3



```
1 #!/bin/bash
2 # calculatrice.sh - Calculatrice simple
3
4 # Définition des nombres
5 nombre1=15
6 nombre2=4
7
8 # Calculs
9 somme=$((nombre1 + nombre2))
10 difference=$((nombre1 - nombre2))
11 produit=$((nombre1 * nombre2))
12 modulo=$((nombre1 % nombre2))
13
14 # Affichage formaté
15 echo "$nombre1 + $nombre2 = $somme"
16 echo "$nombre1 - $nombre2 = $difference"
17 echo "$nombre1 * $nombre2 = $produit"
18 echo "$nombre1 % $nombre2 = $modulo"
19
20 # Bonus : Division simple (attention division par zéro)
21 # Note: Vérification avancée sera vue en séance 2
22 division=$((nombre1 / nombre2))
23 echo "$nombre1 / $nombre2 = $division"
```

# **Exercice 1.4 : Manipulation de Chaînes**

**Objectif :** Travailler avec les chaînes de caractères

**À créer :** chaines.sh

**Fonctionnalités :**

- Stocker prénom et nom séparément
- Concaténer avec espace
- Calculer la longueur
- Convertir en majuscules/minuscules

# Manipulation de Chaînes - Techniques



```
1 chaine="Hello World"
2
3 # Longueur
4 echo ${#chaine}                      # 11
5
6 # Extraction
7 echo ${chaine:0:5}                    # Hello
8 echo ${chaine:6}                      # World
9
10 # Remplacement
11 echo ${chaine/World/Bash}            # Hello Bash
12 echo ${chaine//l/L}                  # HeLLo WorLd
13
14 # Conversion
15 echo ${chaine^^}                    # HELLO WORLD (majuscules)
16 echo ${chaine,,}                    # hello world (minuscules)
```

# Solution Exercice 1.4



```
1 #!/bin/bash
2 # chaines.sh - Manipulation de chaînes
3
4 # Variables
5 prenom="Alice"
6 nom="Dupont"
7
8 # Concaténation
9 nom_complet="${prenom} ${nom}"
10 echo "Nom complet : $nom_complet"
11
12 # Longueur
13 longueur=${#nom_complet}
14 echo "Longueur : $longueur caractères"
15
16 # Conversions
17 echo "Majuscules : ${nom_complet^^}"
18 echo "Minuscules : ${nom_complet,,}"
19
20 # Extraction
21 echo "3 premiers caractères du prénom : ${prenom:0:3}"
22
23 # Bonus : Format nom, prénom
24 echo "Format inversé : ${nom}, ${prenom}"
```

# Pause

 15 minutes

**Après la pause :**

- Entrées utilisateur et paramètres
- Scripts interactifs
- Paramètres positionnels

# **Entrées Utilisateur et Paramètres**

## **(1h30)**

# La Commande read

```
1 #!/bin/bash
2
3 # Lecture simple
4 echo "Quel est votre nom ?"
5 read nom
6 echo "Bonjour $nom !"
7
8 # Lecture avec prompt
9 read -p "Votre âge : " age
10 echo "Vous avez $age ans"
```

**Syntaxe :** read [options] variable

# Options de read



```
1 # Prompt intégré
2 read -p "Nom d'utilisateur : " username
3
4 # Masquer la saisie (mot de passe)
5 read -s -p "Mot de passe : " password
6 echo # Retour à la ligne
7
8 # Timeout (délai d'attente)
9 read -t 10 -p "Répondez dans 10 secondes : " reponse
10
11 # Lecture d'un seul caractère
12 read -n 1 -p "Appuyez sur une touche..." touche
13 echo
```

# Gestion des Timeouts

```
1 #!/bin/bash
2
3 echo "Question avec délai..."
4 echo "Votre couleur préférée (vous avez 5 secondes) :"
5 read -t 5 couleur
6
7 # Vérification simple avec code de retour
8 echo "Code de retour de read : $?"
9 echo "Couleur saisie : $couleur"
10
11 # Note: Gestion avancée avec if/else sera vue en séance 2
```

# Paramètres Positionnels



```
1 #!/bin/bash
2 # script.sh param1 param2 param3
3
4 echo "Nom du script : $0"
5 echo "Premier paramètre : $1"
6 echo "Deuxième paramètre : $2"
7 echo "Troisième paramètre : $3"
8 echo "Nombre de paramètres : $#"
9 echo "Tous les paramètres : $@"
10 echo "Tous les paramètres (chaîne) : $*"
```

**Usage:** ./script.sh arg1 arg2 arg3

# Variables Spéciales

Variable	Description
\$0	Nom du script
\$1, \$2, ...	Paramètres positionnels
\$#	Nombre de paramètres
\$@	Tous les paramètres (tableau)
\$*	Tous les paramètres (chaîne)
\$\$	PID du script
\$?	Code de retour dernière commande

# Utilisation des Paramètres



```
1 #!/bin/bash
2
3 # Affichage d'informations sur les paramètres
4 echo "Nombre de paramètres reçus : $#"
5 echo "Nom du script : $0"
6
7 # Utiliser les paramètres avec message informatif
8 nom=$1
9 age=$2
10
11 echo "Premier paramètre (nom) : $nom"
12 echo "Deuxième paramètre (age) : $age"
13
14 # Affichage final
15 echo "Bonjour $nom, vous avez $age ans"
16
17 # Note: Vérification avancée avec conditions sera vue en séance 2
```

# Shift - Décaler les Paramètres



```
1 #!/bin/bash
2
3 echo "Paramètres initiaux : $@"
4 echo "Premier paramètre : $1"
5
6 # Décaler une fois pour voir l'effet
7 shift
8 echo "Après shift, nouveau \$1 : $1"
9 echo "Paramètres restants : $@"
10 echo "Nouveau nombre de paramètres : $#"
11
12 # Note: Boucle avec shift sera vue en séance 2 (structures de contrôle)
```



# TP Pratique

## Scripts Interactifs avec Paramètres

**Durée :** 45 minutes

# Exercice 1.5 : Salutation Interactive

**Objectif :** Utiliser `read` pour l'interaction utilisateur

**À créer :** `salutation.sh`

**Fonctionnalités :**

- Demander le nom de l'utilisateur
- Demander son âge
- Demander sa ville (avec délai de 10s)
- Afficher un message personnalisé

# Solution Exercice 1.5



```
1 #!/bin/bash
2 # salutation.sh - Script de salutation interactif
3
4 # Demande du nom
5 read -p "Quel est votre nom ? " nom
6
7 # Demande de l'âge
8 read -p "Quel est votre âge ? " age
9
10 # Demande de la ville avec timeout
11 echo "Quelle est votre ville ? (10 secondes pour répondre)"
12 read -t 10 ville
13 echo "Code de retour : $?"
14
15 # Définir une valeur par défaut si vide
16 ville=${ville:-"inconnue"}
17 echo "Ville définie : $ville"
18
19 # Message personnalisé
20 echo ""
21 echo "==== RÉSUMÉ ==="
22 echo "Bonjour $nom !"
23 echo "Vous avez $age ans"
24 echo "Vous habitez à : $ville"
25 echo "Ravi de vous rencontrer !"
```

# **Exercice 1.6 : Gestionnaire de Paramètres**

**Objectif :** Maîtriser les paramètres positionnels

**À créer :** parametres.sh

**Usage :** ./parametres.sh arg1 arg2 arg3

**Fonctionnalités :**

- Afficher le nom du script
- Afficher le nombre de paramètres
- Afficher chaque paramètre individuellement
- Afficher tous les paramètres

# Solution Exercice 1.6



```
1 #!/bin/bash
2 # parametres.sh - Gestionnaire de paramètres
3
4 echo "==== INFORMATIONS SCRIPT ===="
5 echo "Nom du script : $0"
6 echo "Nombre de paramètres reçus : $#"
7 echo ""
8
9 # Affichage d'un message informatif sur les paramètres
10 echo "Note : Ce script fonctionne mieux avec des paramètres"
11 echo "Usage recommandé : $0 <param1> <param2> ..."
12 echo "Exemple : $0 hello world test"
13 echo ""
14
15 echo "==== PARAMÈTRES INDIVIDUELS ===="
16 echo "Premier paramètre (\$1) : $1"
17 echo "Deuxième paramètre (\$2) : $2"
18 echo "Troisième paramètre (\$3) : $3"
19 echo ""
20
21 echo "==== TOUS LES PARAMÈTRES ===="
22 echo "Avec \$@ : $@"
23 echo "Avec \$* : $*"
24 echo ""
25
26 echo "==== PARAMÈTRES UN PAR UN ===="
27 echo "Paramètre 1 : $1"
28 echo "Paramètre 2 : $2"
29 echo "Paramètre 3 : $3"
30 echo "Paramètre 4 : $4"
31 echo "Paramètre 5 : $5"
```

# Exercice Bonus : Script d'Aide



```
1#!/bin/bash
2# parametres.sh - Version avec aide intégrée
3
4# Fonction simple d'aide (concept de base)
5# Note: Fonctions avancées et tests conditionnels seront vus en séance 2
6
7echo "==== AIDE DU SCRIPT ==="
8echo "Usage : $0 <paramètres>"
9echo ""
10echo "EXEMPLES :"
11echo "  $0 hello world"
12echo "  $0 test1 test2 test3"
13echo ""
14echo "Ce script affiche des informations sur les paramètres reçus"
```

# Récapitulatif Matinée

- ✓ **Introduction au Shell et Bash** : Historique, rôle, variables d'environnement
- ✓ **Structure d'un script** : Shebang, permissions, commentaires, bonnes pratiques
- ✓ **Variables et types** : Déclaration, chaînes, nombres, opérations
- ✓ **Entrées utilisateur** : read, options, paramètres positionnels
- ✓ **Pratique** : 4 exercices complets réalisés

# Pause Déjeuner

 12h00 - 13h00

**Cet après-midi :**

- Variables et calculs avancés
- Entrées utilisateur avancées
- Exercices pratiques complémentaires

# **Séance 1 - Après-midi**

## **Variables Avancées et Calculs**

### **(3h30)**

# Objectifs de l'Après-midi

- Approfondir la manipulation des variables
- Maîtriser les calculs arithmétiques avancés
- Créer des scripts plus complexes
- Consolider les acquis avec des exercices pratiques

# **Variables Avancées**

## **Techniques de manipulation**

# Variables d'Environnement vs Variables Shell



```
1 # Variable shell (locale au script)
2 ma_variable="locale"
3 echo "Variable locale : $ma_variable"
4
5 # Variable d'environnement (exportée)
6 export MA_VARIABLE_GLOBALE="globale"
7 echo "Variable exportée : $MA_VARIABLE_GLOBALE"
8
9 # Afficher les variables d'environnement
10 echo "Utilisateur système : $USER"
11 echo "Répertoire personnel : $HOME"
12
13 # Variable avec valeur par défaut (substitution)
14 echo "Utilisateur : ${USER:-utilisateur_inconnu}"
15 echo "Shell : ${SHELL:-shell_inconnu}"
16
17 # Note: Tests de vérification avec if seront vus en séance 2
```

# Substitution de Variables



```
1 variable="hello world"
2
3 # Substitution avec valeur par défaut
4 echo ${variable:-"valeur par défaut"}
5
6 # Substitution avec assignation par défaut
7 echo ${nouvelle_var:="valeur assignée"}
8
9 # Substitution avec message d'erreur
10 echo ${obligatoire:?Cette variable est requise}
11
12 # Substitution avec valeur alternative
13 echo ${variable:+variable est définie}
```

# Manipulation Avancée de Chaînes



```
1 fichier="document.txt"
2
3 # Suppression du suffixe
4 echo ${fichier%.txt}          # document
5 echo ${fichier%.*}            # document
6
7 # Suppression du préfixe
8 chemin="/home/user/document.txt"
9 echo ${chemin##*/}             # document.txt
10 echo ${chemin##*/}            # home/user/document.txt
11
12 # Remplacement
13 echo ${fichier/txt/pdf}      # document.pdf
14 echo ${fichier//t/T}          # document.TxT
```

# Tableaux Simples



```
1 # Déclaration d'un tableau
2 fruits=("pomme" "banane" "orange" "kiwi")
3
4 # Accès aux éléments
5 echo ${fruits[0]}                      # pomme
6 echo ${fruits[2]}                      # orange
7
8 # Tous les éléments
9 echo ${fruits[@]}                      # tous les fruits
10 echo ${fruits[*]}                      # tous les fruits (chaîne)
11
12 # Nombre d'éléments
13 echo ${#fruits[@]}                     # 4
14
15 # Ajouter un élément
16 fruits[4]="mangue"
17 fruits+=("ananas")
```

# Calculs Arithmétiques Avancés



```
1 # Méthodes de calcul
2 a=10
3 b=3
4
5 # Méthode 1 : $(( ))
6 resultat=$((a * b + 5))
7
8 # Méthode 2 : let
9 let "resultat = a * b + 5"
10
11 # Méthode 3 : declare -i
12 declare -i nombre=10
13 nombre=nombre*2      # Calcul automatique
14
15 # Opérations bit à bit
16 echo $((5 & 3))      # ET binaire
17 echo $((5 | 3))      # OU binaire
18 echo $((5 ^ 3))      # XOR binaire
```

# Calculs avec Virgule Flottante



```
1 # Limitation de Bash : seulement nombres entiers
2 a=10
3 b=3
4 division_entiere=$((a / b))
5 echo "Division entière : $division_entiere" # 3
6
7 # Pour calculs précis, utiliser bc (calculatrice en ligne de commande)
8 resultat=$(echo "scale=2; 10/3" | bc)
9 echo "Division précise : $resultat" # 3.33
10
11 # Autre exemple avec bc
12 calcul=$(echo "scale=2; 15.5 * 2.1" | bc)
13 echo "Multiplication décimale : $calcul"
14
15 # Note: bc sera approfondi en séance 3 (manipulation de texte)
16 # Note: Fonctions seront vues en séance 2
```

# Variables de Contrôle



```
1 # Variables pour configuration (concepts de base)
2 debug="true"
3 verbose="false"
4
5 echo "Mode debug : $debug"
6 echo "Mode verbose : $verbose"
7
8 # Note: Tests conditionnels avec if seront vus en séance 2
```

# Codes de Couleur dans le Terminal



```
1 # Définition des codes de couleur ANSI
2 RED='\033[0;31m'      # Rouge
3 GREEN='\033[0;32m'     # Vert
4 BLUE='\033[0;34m'      # Bleu
5 YELLOW='\033[1;33m'    # Jaune
6 NC='\033[0m'           # No Color (réinitialise)
7
8 # Utilisation avec echo -e (interprète les séquences d'échappement)
9 echo -e "${RED}Erreur${NC} : Quelque chose ne va pas"
10 echo -e "${GREEN}Succès${NC} : Opération réussie"
11 echo -e "${BLUE}Information${NC} : Message informatif"
12
13 # Pourquoi les codes fonctionnent :
14 # \033 = caractère d'échappement ESC
15 # [0;31m = séquence ANSI pour rouge
16 # \033[0m = réinitialise la couleur
```



# TP Pratique Avancé

## Calculatrice et Utilitaires

Durée : 1h30

# **Exercice 1.7 : Calculatrice Avancée**

**Objectif :** Créer une calculatrice interactive complète

**À créer :** `calculatrice_avancee.sh`

**Fonctionnalités :**

- Menu interactif
- Opérations de base et avancées
- Historique des calculs
- Gestion des erreurs

# Structure Calculatrice Avancée



```
1 #!/bin/bash
2 # calculatrice_avancee.sh
3
4 # Couleurs
5 RED='\033[0;31m'
6 GREEN='\033[0;32m'
7 BLUE='\033[0;34m'
8 YELLOW='\033[1;33m'
9 NC='\033[0m'
10
11 # Tableau pour historique
12 declare -a historique
13
14 # Menu simple (sans fonction et sans boucle interactive)
15 clear
16 echo -e "${BLUE}==> CALCULATRICE SÉANCE 1 ==>${NC}"
17 echo "Ce programme effectue des calculs simples"
18 echo ""
19 echo "Opérations disponibles :"
20 echo "- Addition, soustraction, multiplication, division"
21 echo "- Avec deux nombres prédéfinis"
22 echo ""
23 echo "Note: Menu interactif et choix utilisateur en séance 2"
```

# Solution Partielle Calculatrice



```
1 # Version simplifiée pour séance 1 - opérations de base
2 operation="addition"
3 a=10
4 b=5
5
6 # Calculs directs (sans structures de contrôle)
7 addition=$((a + b))
8 soustraction=$((a - b))
9 multiplication=$((a * b))
10 division=$((a / b))
11
12 echo "==== CALCULATRICE SIMPLE ==="
13 echo "$a + $b = $addition"
14 echo "$a - $b = $soustraction"
15 echo "$a * $b = $multiplication"
16 echo "$a / $b = $division"
17
18 # Note: Structures case, fonctions et gestion d'erreurs en séance 2
```

# **Exercice 1.8 : Gestionnaire de Fichiers**

**Objectif :** Créer un utilitaire de gestion de fichiers

**À créer :** `file_manager.sh`

**Fonctionnalités :**

- Lister les fichiers avec détails
- Calculer la taille des répertoires
- Rechercher des fichiers
- Statistiques d'utilisation

# Solution File Manager



```
1 #!/bin/bash
2 # file_manager.sh - Gestionnaire de fichiers
3
4 # Gestionnaire de fichiers simplifié - Version séance 1
5 répertoire="."
6
7 echo "==== INFORMATIONS SUR LE RÉPERTOIRE COURANT ==="
8
9 # Utilisation de commandes simples
10 echo "Répertoire analysé : $(pwd)"
11 echo ""
12
13 # Listage simple
14 echo "==== CONTENU DU RÉPERTOIRE ==="
15 ls -la
16
17 echo ""
18 echo "==== STATISTIQUES SIMPLES ==="
19
20 # Comptage avec des commandes directes
21 echo "Nombre total d'éléments : $(ls -1 | wc -l)"
22 echo "Espace disque utilisé :"
23 du -sh .
24
25 # Note: Fonctions, boucles et recherche avancée en séance 2
```

# Redirections de Base

```
● ● ●

1 # Redirection de la sortie vers un fichier
2 echo "Hello World" > fichier.txt          # Écrase le fichier
3 echo "Nouvelle ligne" >> fichier.txt      # Ajoute au fichier
4
5 # Exemples pratiques
6 date > horodatage.txt                      # Date dans un fichier
7 ls -la >> liste_fichiers.txt                # Ajoute la liste
8
9 # Redirection des erreurs
10 ls fichier_inexistant 2> erreurs.txt       # Erreurs dans un fichier
11 ls fichier_inexistant 2>> erreurs.txt      # Ajoute les erreurs
12
13 # Tout rediriger (sortie + erreurs)
14 commande &> tout.txt                       # Bash moderne
15 commande > tout.txt 2>&1                   # Compatible partout
16
17 # Ignorer complètement
18 commande > /dev/null 2>&1                  # Silence total
```

**Note :** Redirections avancées et pipes seront approfondis en séance 3

# **Exercice 1.9 : Système de Logs**

**Objectif :** Créer un système de logging réutilisable

**À créer :** logger.sh

**Fonctionnalités :**

- Niveaux de log (DEBUG, INFO, WARN, ERROR)
- Rotation des logs
- Formatage avec timestamps
- Configuration par variables

# Solution Logger System



```
1 #!/bin/bash
2 # logger.sh - Système de logging
3
4 # Configuration
5 LOG_DIR="/tmp/mes_logs"
6 LOG_FILE="$LOG_DIR/application.log"
7 MAX_LOG_SIZE=1048576 # 1MB en bytes
8
9 # Niveaux de log
10 DEBUG=0
11 INFO=1
12 WARN=2
13 ERROR=3
14
15 LOG_LEVEL=$INFO # Niveau par défaut
16
17 # Système de log simple - Version séance 1
18
19 # Configuration de base
20 LOG_FILE="/tmp/mon_script.log"
21 timestamp=$(date '+%Y-%m-%d %H:%M:%S')
22
23 # Création du répertoire si nécessaire
24 mkdir -p /tmp
25
26 # Fonction de log simple (sans conditions ni tableaux associatifs)
27 message="Script démarré"
28 echo "[${timestamp}] [INFO] $message" >> "$LOG_FILE"
29 echo "Log écrit dans : $LOG_FILE"
30
31 # Affichage avec couleur simple
```

# Intégration des Outils



```
1 #!/bin/bash
2 # script_principal.sh - Utilisation des outils créés
3
4 # Utilisation simple du logging - Version séance 1
5
6 # Variables de base
7 fichier="test.txt"
8 LOG_FILE="/tmp/mon_script.log"
9
10 # Log simple du démarrage
11 echo "[$(date '+%Y-%m-%d %H:%M:%S')] [INFO] Démarrage de l'application" >
12
13 # Exemple d'utilisation dans le script
14 echo "Vérification du fichier : $fichier"
15
16 # Test simple d'existence (sans if complexe)
17 ls "$fichier" 2>/dev/null && echo "Fichier trouvé" || echo "Fichier non t
18
19 # Log de fin
20 echo "[$(date '+%Y-%m-%d %H:%M:%S')] [INFO] Fin du traitement" >> "$LOG_F
21
22 # Note: Fonctions, tests avancés et trap en séance 2
```

# Récapitulatif Séance 1 Complète

## Matinée :

- Introduction au Shell et Bash
- Structure des scripts et bonnes pratiques
- Variables de base et manipulation de chaînes
- Entrées utilisateur et paramètres

## Après-midi :

- Variables avancées et tableaux
- Calculs arithmétiques complexes
- Outils réutilisables (calculatrice, file manager, logger)

# Compétences Acquises

- 🎯 **Créer et exécuter des scripts Bash**
- 🎯 **Gérer les variables et types de données**
- 🎯 **Interagir avec l'utilisateur**
- 🎯 **Manipuler les chaînes et effectuer des calculs**
- 🎯 **Structurer du code réutilisable**

# **Préparation Séance 2**

**Prochaine séance : Structures de contrôle**

**À réviser :**

- Variables et leur utilisation
- Paramètres de scripts
- Commandes de base vues aujourd'hui

**À préparer :**

- Avoir un environnement de test prêt
- Scripts de la séance 1 sauvegardés

**Questions Finales ?**

**Merci pour votre participation !**

# **Formation Bash Scripting - Séance 2**

**Formation Bash Scripting -  
Séance 2**

**Structures de Contrôle**

**7 heures - Matinée et Après-midi**

*Niveau BAC+2 TSSR - 35h de formation*

# Rappel Séance 1

- Variables et types de données**
- Entrées utilisateur et paramètres**
- Structure des scripts**
- Manipulation de chaînes**
- Calculs arithmétiques**

**Aujourd'hui :** Nous allons apprendre à contrôler le flux d'exécution !

# Objectifs de la Séance 2

- Maîtriser les **conditions** et tests
- Utiliser les **structures de contrôle** avancées
- Implémenter des **boucles** efficaces
- Créer et utiliser des **fonctions**
- Développer des scripts **robustes** et **modulaires**

# **Séance 2 - Matinée**

## **Conditions et Tests**

**(3h30)**

# **Conditions et Tests**

## **(2h)**

# Structure if/then/else/fi



```
1 #!/bin/bash
2
3 age=18
4
5 if [ $age -ge 18 ]; then
6     echo "Vous êtes majeur"
7 else
8     echo "Vous êtes mineur"
9 fi
```

## Syntaxe :



```
1 if [ condition ]; then
2     # commandes si vrai
3 else
4     # commandes si faux
5 fi
```

# Conditions Multiples



```
1 #!/bin/bash
2
3 age=25
4
5 if [ $age -lt 18 ]; then
6     echo "Mineur"
7 elif [ $age -lt 65 ]; then
8     echo "Adulte"
9 else
10    echo "Senior"
11 fi
```

**Structure:** if / elif / else / fi

# Opérateurs de Comparaison Numériques

Opérateur	Description	Exemple
-eq	Égal à	[ \$a -eq \$b ]
-ne	Différent de	[ \$a -ne \$b ]
-lt	Inférieur à	[ \$a -lt \$b ]
-le	Inférieur ou égal	[ \$a -le \$b ]
-gt	Supérieur à	[ \$a -gt \$b ]
-ge	Supérieur ou égal	[ \$a -ge \$b ]

# Opérateurs de Comparaison de Chaînes

Opérateur	Description	Exemple
=	Égal à	[ "\$str1" = "\$str2" ]
!=	Différent de	[ "\$str1" != "\$str2" ]
<	Inférieur (ordre alphabétique)	[ [ "\$str1" < "\$str2" ] ]
>	Supérieur (ordre alphabétique)	[ [ "\$str1" > "\$str2" ] ]
-z	Chaîne vide	[ -z "\$str" ]
-n	Chaîne non vide	[ -n "\$str" ]

# Tests sur les Fichiers

Test	Description	Exemple
-f	Fichier existe et est régulier	[ -f "fichier.txt" ]
-d	Répertoire existe	[ -d "/home/user" ]
-e	Fichier/répertoire existe	[ -e "element" ]
-r	Lisible	[ -r "fichier.txt" ]
-w	Écriture possible	[ -w "fichier.txt" ]
-x	Exécutable	[ -x "script.sh" ]

# Exemple Tests de Fichiers



```
1 #!/bin/bash
2
3 fichier="document.txt"
4
5 if [ -f "$fichier" ]; then
6     echo "Le fichier existe"
7
8     if [ -r "$fichier" ]; then
9         echo "Fichier lisible"
10    fi
11
12    if [ -w "$fichier" ]; then
13        echo "Fichier modifiable"
14    fi
15
16    if [ -s "$fichier" ]; then
17        echo "Fichier non vide"
18    else
19        echo "Fichier vide"
20    fi
21 else
22     echo "Le fichier n'existe pas"
23 fi
```

# Opérateurs Logiques



```
1 #!/bin/bash
2
3 age=25
4 nom="Alice"
5
6 # ET logique (&&)
7 if [ $age -ge 18 ] && [ -n "$nom" ]; then
8     echo "Utilisateur valide"
9 fi
10
11 # OU logique (||)
12 if [ $age -lt 18 ] || [ -z "$nom" ]; then
13     echo "Données incomplètes"
14 fi
15
16 # NON logique (!)
17 if [ ! -f "fichier.txt" ]; then
18     echo "Fichier inexistant"
19 fi
```

# Différence entre [ ] et [[ ]]



```
1 # Test simple avec [ ]
2 if [ "$var" = "test" ]; then
3     echo "Simple test"
4 fi
5
6 # Test avancé avec [[ ]]
7 if [[ $var == test* ]]; then # Pattern matching
8     echo "Commence par 'test'"
9 fi
10
11 if [[ $var =~ ^[0-9]+$ ]]; then # Regex
12     echo "Nombre entier"
13 fi
14
15 # Opérateurs logiques intégrés
16 if [[ $age -gt 18 && $nom != "" ]]; then
17     echo "Conditions multiples"
18 fi
```



# TP Pratique

## Vérification de Fichiers et Droits

**Durée :** 45 minutes

# **Exercice 2.1 : Vérificateur d'Âge et Permissions**

**Objectif :** Utiliser les conditions pour des vérifications

**À créer :** verificateur.sh

**Fonctionnalités :**

- Demander l'âge de l'utilisateur
- Catégoriser : mineur, majeur, senior (65+)
- Prendre un nom de fichier en paramètre
- Vérifier permissions du fichier
- Rapport complet des vérifications

# Solution Exercice 2.1



```
1 #!/bin/bash
2 # vérificateur.sh - Vérificateur d'âge et permissions
3
4 # Couleurs pour l'affichage
5 RED='\033[0;31m'
6 GREEN='\033[0;32m'
7 BLUE='\033[0;34m'
8 YELLOW='\033[1;33m'
9 NC='\033[0m'
10
11 echo -e "${BLUE}== VÉRIFICATEUR D'ÂGE ET PERMISSIONS ==${NC}"
12
13 # Vérification de l'âge
14 read -p "Quel est votre âge ? " age
15
16 # Validation de l'entrée
17 if ! [[ $age =~ ^[0-9]+$ ]]; then
18     echo -e "${RED}Erreur : Veuillez entrer un nombre valide${NC}"
19     exit 1
20 fi
21
22 # Catégorisation par âge
23 echo -e "\n${YELLOW}== CATÉGORIE D'ÂGE ==${NC}"
24 if [ $age -lt 18 ]; then
25     echo -e "${YELLOW}Statut : Mineur ($age ans)${NC}"
26 elif [ $age -lt 65 ]; then
27     echo -e "${GREEN}Statut : Majeur ($age ans)${NC}"
28 else
29     echo -e "${BLUE}Statut : Senior ($age ans)${NC}"
30 fi
```

# **Exercice 2.2 : Analyseur de Mots de Passe**

**Objectif :** Combiner plusieurs conditions

**À créer :** motdepasse.sh

**Fonctionnalités :**

- Demander un mot de passe (masqué)
- Vérifier la longueur (min 8 caractères)
- Vérifier présence chiffres et lettres
- Score de sécurité (Faible/Moyen/Fort)
- Recommandations d'amélioration

# Solution Exercice 2.2



```
1 #!/bin/bash
2 # motdepasse.sh - Analyseur de mots de passe
3
4 # Couleurs
5 RED='\033[0;31m'
6 GREEN='\033[0;32m'
7 YELLOW='\033[1;33m'
8 BLUE='\033[0;34m'
9 NC='\033[0m'
10
11 echo -e "${BLUE}== ANALYSEUR DE MOTS DE PASSE ==${NC}"
12
13 # Demande du mot de passe
14 read -s -p "Entrez votre mot de passe : " password
15 echo # Retour à la ligne
16
17 # Variables de score
18 score=0
19 recommandations=()
20
21 echo -e "\n${YELLOW}== ANALYSE DU MOT DE PASSE ==${NC}"
22
23 # Vérification longueur
24 longueur=${#password}
25 echo "Longueur : $longueur caractères"
26
27 if [ $longueur -ge 8 ]; then
28     echo -e "${GREEN}✓ Longueur suffisante${NC}"
29     ((score += 2))
```

# Solution Exercice 2.2



```
1  if [ $longueur -ge 12 ]; then
2      echo -e "${GREEN}✓ Longueur excellente${NC}"
3      ((score += 1))
4  fi
5 else
6     echo -e "${RED}✗ Trop court (minimum 8 caractères)${NC}"
7     recommandations+=("Utilisez au moins 8 caractères")
8 fi
9
10 # Vérification présence de chiffres
11 if [[ $password =~ [0-9] ]]; then
12     echo -e "${GREEN}✓ Contient des chiffres${NC}"
13     ((score += 1))
14 else
15     echo -e "${RED}✗ Aucun chiffre${NC}"
16     recommandations+=("Ajoutez des chiffres")
17 fi
18
19 # Vérification présence de lettres
20 if [[ $password =~ [a-zA-Z] ]]; then
21     echo -e "${GREEN}✓ Contient des lettres${NC}"
22     ((score += 1))
23 else
24     echo -e "${RED}✗ Aucune lettre${NC}"
25     recommandations+=("Ajoutez des lettres")
26 fi
27
28 # Vérification minuscules et majuscules
29 if [[ $password =~ [a-z] ]] && [[ $password =~ [A-Z] ]]; then
```

# Solution Exercice 2.2



```
1     echo -e "${GREEN}✓ Mix majuscules/minuscules${NC}"
2     ((score += 1))
3 else
4     echo -e "${YELLOW}⚠ Utilisez majuscules ET minuscules${NC}"
5     recommandations+=("Mélangez majuscules et minuscules")
6 fi
7
8 # Bonus : Caractères spéciaux
9 if [[ $password =~ [^a-zA-Z0-9] ]]; then
10    echo -e "${GREEN}✓ Contient des caractères spéciaux${NC}"
11    ((score += 2))
12 else
13    echo -e "${YELLOW}⚠ Aucun caractère spécial${NC}"
14    recommandations+=("Ajoutez des caractères spéciaux (!@#$%)")
15 fi
16
17 # Évaluation du score
18 echo -e "\n${YELLOW}--- ÉVALUATION ---${NC}"
19 echo "Score : $score/7"
20
21 if [ $score -le 2 ]; then
22    echo -e "${RED}🔴 Sécurité : FAIBLE${NC}"
23 elif [ $score -le 4 ]; then
24    echo -e "${YELLOW}🟡 Sécurité : MOYENNE${NC}"
25 else
26    echo -e "${GREEN}🟢 Sécurité : FORTE${NC}"
27 fi
28
```

# Solution Exercice 2.2



```
1
2 # Recommandations
3 if [ ${#recommandations[@]} -gt 0 ]; then
4     echo -e "\n${YELLOW}== RECOMMANDATIONS ==${NC}"
5     for rec in "${recommandations[@]}"; do
6         echo -e "${YELLOW}• $rec${NC}"
7     done
8 else
9     echo -e "\n${GREEN}🎉 Excellent mot de passe !${NC}"
10 fi
```

# **Conditions Avancées**

## **(1h30)**

# Structure case/esac

```
1 #!/bin/bash
2
3 read -p "Choisissez une option (1-4) : " choix
4
5 case $choix in
6     1)
7         echo "Option 1 sélectionnée"
8         ;;
9     2)
10        echo "Option 2 sélectionnée"
11        ;;
12    3|4) # Plusieurs valeurs
13        echo "Option 3 ou 4 sélectionnée"
14        ;;
15    *) # Cas par défaut
16        echo "Option invalide"
17        ;;
18 esac
```

# Patterns dans case



```
1 #!/bin/bash
2
3 read -p "Entrez un mot : " mot
4
5 case $mot in
6     [Bb]onjour)
7         echo "Salutation détectée"
8         ;;
9     [0-9])
10        echo "Un seul chiffre"
11        ;;
12    [0-9][0-9])
13        echo "Deux chiffres"
14        ;;
15    test*)
16        echo "Commence par 'test'"
17        ;;
18    *[Pp]ython*)
19        echo "Contient 'python'"
20        ;;
21    *.txt)
22        echo "Fichier texte"
23        ;;
24    *)
25        echo "Autre pattern"
26        ;;
27 esac
```

# Fonctions de Base

## (1h)

# Déclaration et Appel de Fonctions

```
1 #!/bin/bash
2
3 # Déclaration d'une fonction simple
4 dire_bonjour() {
5     echo "Bonjour tout le monde !"
6 }
7
8 # Appel de la fonction
9 dire_bonjour
10
11 # Fonction avec paramètres
12 saluer() {
13     local nom=$1
14     local age=$2
15     echo "Bonjour $nom, vous avez $age ans"
16 }
17
18 # Appel avec paramètres
19 saluer "Alice" 25
20 saluer "Bob" 30
```

**Syntaxe :** nom\_fonction( ) { commandes; }

# Variables Locales et Paramètres



```
1 #!/bin/bash
2
3 # Fonction avec variables locales
4 calculer_somme() {
5     local a=$1          # Premier paramètre
6     local b=$2          # Deuxième paramètre
7     local somme=$((a + b)) # Variable locale
8
9     echo "Somme : $somme"
10 }
11
12 # Utilisation
13 calculer_somme 15 25
14
15 # Fonction pour afficher un menu
16 afficher_menu() {
17     echo "==== MENU PRINCIPAL ===="
18     echo "1. Option 1"
19     echo "2. Option 2"
20     echo "3. Quitter"
21     echo -n "Votre choix : "
22 }
23
24 afficher_menu
```

**Important :** `local` limite la variable à la fonction

# Valeurs de Retour

```
● ● ●
```

```
1 #!/bin/bash
2
3 # Retour par echo (capture de sortie)
4 additionner() {
5     local a=$1
6     local b=$2
7     echo $((a + b))
8 }
9
10 # Retour par code de retour (0 = succès, 1+ = erreur)
11 fichier_existe() {
12     local fichier=$1
13     if [ -f "$fichier" ]; then
14         return 0    # Succès
15     else
16         return 1    # Échec
17     fi
18 }
19
20 # Utilisation
21 resultat=$(additionner 5 3)
22 echo "5 + 3 = $resultat"
23
24 if fichier_existe "/etc/passwd"; then
25     echo "Le fichier existe"
26 fi
```

**Note :** Fonctions avancées seront vues en fin de

# Tests avec Expressions Régulières



```
1 #!/bin/bash
2
3 # Variables pour les tests
4 email="user@example.com"
5 telephone="0123456789"
6
7 # Validation d'email avec regex (sans fonction)
8 if [[ $email =~ ^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$ ]]; then
9     echo "Email '$email' est valide"
10 else
11     echo "Email '$email' est invalide"
12 fi
13
14 # Validation numéro de téléphone français
15 if [[ $telephone =~ ^0[1-9]([0-9]{8})$ ]]; then
16     echo "Numéro '$telephone' est valide"
17 else
18     echo "Numéro invalide (format : 0123456789)"
19 fi
20
21 # Test avec différentes valeurs
22 email_invalide="email-sans-arobase"
23 if [[ $email_invalide =~ ^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$ ]]; then
24     echo "Email '$email_invalide' est valide"
25 else
26     echo "Email '$email_invalide' est invalide"
27 fi
28
29 # Note: Validation avec fonctions sera vue plus tard dans cette séance
```

# Tests Multiples et Imbriqués



```
1 #!/bin/bash
2
3 # Variables pour les tests
4 nom="Alice"
5 age=25
6 email="alice@example.com"
7
8 # Tests imbriqués (sans fonction)
9 if [ -n "$nom" ]; then
10    echo "Nom présent : $nom"
11    if [ $age -ge 18 ]; then
12        echo "Utilisateur majeur : $age ans"
13        if [[ $email =~ @ ]]; then
14            echo "Email valide : $email"
15            echo "✅ Utilisateur complètement valide"
16        else
17            echo "❌ Email invalide : $email"
18        fi
19    else
20        echo "❌ Utilisateur mineur : $age ans"
21    fi
22 else
23    echo "❌ Nom manquant"
24 fi
25
26 # Test avec && (plus concis)
27 if [ -n "$nom" ] && [ $age -ge 18 ] && [[ $email =~ @ ]]; then
28    echo "Validation rapide réussie"
29 fi
```



# TP Pratique

Menu Interactif avec case

Durée : 45 minutes

# **Exercice 2.3 : Menu Système Interactif**

**Objectif :** Créer un menu avec case/esac

**À créer :** menu.sh

**Options du menu :**

1. Afficher la date et l'heure
2. Afficher l'espace disque
3. Afficher les utilisateurs connectés
4. Afficher les processus
5. Quitter

Le menu doit se répéter jusqu'à "Quitter"

# Solution Exercice 2.3

```
1 #!/bin/bash
2 # menu.sh - Menu système interactif
3
4 # Couleurs
5 RED='\033[0;31m'
6 GREEN='\033[0;32m'
7 BLUE='\033[0;34m'
8 YELLOW='\033[1;33m'
9 CYAN='\033[0;36m'
10 NC='\033[0m'
11
12 afficher_menu() {
13     clear
14     echo -e "${BLUE}===== ${NC}"
15     echo -e "${BLUE}      MENU SYSTÈME INTERACTIF      ${NC}"
16     echo -e "${BLUE}===== ${NC}"
17     echo
18     echo -e "${CYAN}1.${NC} Afficher la date et l'heure"
19     echo -e "${CYAN}2.${NC} Afficher l'espace disque"
20     echo -e "${CYAN}3.${NC} Afficher les utilisateurs connectés"
21     echo -e "${CYAN}4.${NC} Afficher les processus"
22     echo -e "${CYAN}5.${NC} Informations système complètes"
23     echo -e "${RED}6.${NC} Quitter"
24     echo
25     echo -n -e "${YELLOW}Votre choix [1-6] : ${NC}"
26 }
27
```

# Solution Exercice 2.3



```
1 attendre_touche() {
2     echo
3     echo -n -e "${YELLOW}Appuyez sur Entrée pour continuer...${NC}"
4     read
5 }
```

# Solution Exercice 2.3



```
1
2 # Fonction principale du menu
3 gerer_menu() {
4     afficher_menu
5     read choix
6
7     case $choix in
8         1)
9             echo -e "\n${GREEN}--- DATE ET HEURE ---${NC}"
10            echo -e "${CYAN}Date courte :${NC} $(date '+%d/%m/%Y')"
11            echo -e "${CYAN}Heure :${NC} $(date '+%H:%M:%S')"
12            echo -e "${CYAN}Date complète :${NC} $(date '+%A %d %B %Y à %H:%M:%S')"
13            echo -e "${CYAN}Timestamp :${NC} $(date '+%s')"
14            attendre_touche
15            ;;
16        2)
17            echo -e "\n${GREEN}--- ESPACE DISQUE ---${NC}"
18            echo -e "${CYAN}Utilisation par système de fichiers :${NC}"
19            df -h | head -1 # En-tête
20            df -h | grep -E '^/dev/' | head -5 # 5 premiers disques
21            echo
22            echo -e "${CYAN}Espace total utilisé :${NC}"
23            df -h --total | tail -1
24            attendre_touche
25            ;;
```

# Solution Exercice 2.3

```
1 3)
2         echo -e "\n${GREEN}==== UTILISATEURS CONNECTÉS ===${NC}"
3         echo -e "${CYAN}Utilisateurs actuellement connectés :${NC}"
4         who
5         echo
6         echo -e "${CYAN}Nombre d'utilisateurs :${NC} $(who | wc -l)"
7         echo
8         echo -e "${CYAN}Historique des connexions (5 dernières) :${NC}"
9         last -5
10        attendre_touche
11        ;;
12
13        4)
14         echo -e "\n${GREEN}==== PROCESSUS ===${NC}"
15         echo -e "${CYAN}Top 10 des processus (CPU) :${NC}"
16         ps aux --sort=-%cpu | head -11
17         echo
18         echo -e "${CYAN}Nombre total de processus :${NC} $(ps aux | wc -l)"
19         echo -e "${CYAN}Processus de l'utilisateur actuel :${NC} $(ps aux | grep $(id -u) | head -1 | awk '{print $1 }')"
20         attendre_touche
21        ;;
22
23        5)
24         echo -e "\n${GREEN}==== INFORMATIONS SYSTÈME COMPLÈTES ===${NC}"
25         echo -e "${CYAN}Hostname :${NC} $(hostname)"
26         echo -e "${CYAN}Utilisateur :${NC} $(whoami)"
27         echo -e "${CYAN}Système :${NC} $(uname -s)"
28         echo -e "${CYAN}Version :${NC} $(uname -r)"
29         echo -e "${CYAN}Architecture :${NC} $(uname -m)"
30         echo -e "${CYAN}Uptime :${NC} $(uptime -p)"
31         echo -e "${CYAN}Load average :${NC} $(uptime | awk -F'load average' '{print $2}')"
```

# Solution Exercice 2.3



```
1 if command -v free >/dev/null 2>&1; then
2     echo -e "${CYAN}Mémoire :${NC}"
3     free -h
4 fi
5 attendre_touche
6 ;;
7 6|q|Q|quit|exit)
8     echo -e "\n${GREEN}Au revoir ! 🖐 ${NC}"
9     exit 0
10 ;;
11 *)
12     echo -e "\n${RED}✗ Option invalide ! Veuillez choisir entre"
13     sleep 2
14 ;;
15 esac
16 }
17
18 # Appel de la fonction principale
19 echo "==== DÉMONSTRATION DU MENU ==="
20 echo "Note: Ce menu s'exécute une fois. Version avec boucle en fin de séa
21 gerer_menu
```

# Exercice 2.4 : Classificateur de Fichiers

**Objectif :** Utiliser case avec des patterns

**À créer :** classifier.sh

**Classifications :**

- \*.txt, \*.md → "Fichier texte"
- \*.jpg, \*.png → "Image"
- \*.sh, \*.py → "Script"
- \*.tar, \*.zip → "Archive"
- Autres → "Type inconnu"

**Bonus :** Ajouter la taille du fichier s'il existe

# Solution Exercice 2.4

```
1 #!/bin/bash
2 # classificateur.sh - Classificateur de fichiers
3
4 # Couleurs
5 RED='\033[0;31m'
6 GREEN='\033[0;32m'
7 BLUE='\033[0;34m'
8 YELLOW='\033[1;33m'
9 PURPLE='\033[0;35m'
10 CYAN='\033[0;36m'
11 NC='\033[0m'
12
13 # Vérification des paramètres
14 if [ $# -eq 0 ]; then
15     echo -e "${RED}Usage : $0 <fichier1> [fichier2] ...${NC}"
16     echo -e "${YELLOW}Exemple : $0 document.txt image.jpg script.sh${NC}"
17     exit 1
18 fi
19
20 classifier_fichier() {
21     local fichier=$1
22     local nom_fichier=$(basename "$fichier")
23
24     echo -e "\n${CYAN}== Analyse de : $nom_fichier ===${NC}"
25
26     # Vérifier si le fichier existe
27     if [ ! -e "$fichier" ]; then
28         echo -e "${RED}✗ Fichier inexistant${NC}"
29         return 1
30     fi
31 }
```

# Solution Exercice 2.4

```
1 # Classification selon l'extension
2 case "$nom_fichier" in
3     *.txt|*.md|*.rst|*.doc|*.rtf)
4         echo -e "${GREEN}📄 Type : Fichier texte${NC}"
5         icone="📄"
6         ;;
7     *.jpg|*.jpeg|*.png|*.gif|*.bmp|*.svg|*.webp)
8         echo -e "${PURPLE}🖼️ Type : Image${NC}"
9         icone="🖼️"
10        ;;
11    *.sh|*.py|*.pl|*.rb|*.js|*.php)
12        echo -e "${YELLOW}⚙️ Type : Script${NC}"
13        icone="⚙️"
14        ;;
15    *.tar|*.tar.gz|*.zip|*.rar|*.7z|*.gz)
16        echo -e "${BLUE}📦 Type : Archive${NC}"
17        icone="📦"
18        ;;
19    *.pdf)
20        echo -e "${RED}📕 Type : Document PDF${NC}"
21        icone="📕"
22        ;;
23    *.mp3|*.wav|*.flac|*.ogg)
24        echo -e "${CYAN}🎵 Type : Fichier audio${NC}"
25        icone="🎵"
26        ;;
27    *.mp4|*.avi|*.mkv|*.mov)
28        echo -e "${YELLOW}🎬 Type : Fichier vidéo${NC}"
29        icone="🎬"
30        ;;
```

# Solution Exercice 2.4

```
1      *.html|*.htm|*.css|*.xml)
2          echo -e "${GREEN}🌐 Type : Fichier web${NC}"
3          icone="🌐"
4          ;;
5      *.log)
6          echo -e "${YELLOW}📝 Type : Fichier de log${NC}"
7          icone="📝"
8          ;;
9      .*)
10         echo -e "${CYAN}⚙️ Type : Fichier de configuration${NC}"
11         icone="⚙️"
12         ;;
13     *)
14         echo -e "${RED} ? Type : Inconnu${NC}"
15         icone=" ?"
16         ;;
17     esac
18
19 # Informations détaillées
20 if [ -f "$fichier" ]; then
21     # Taille du fichier
22     taille=$(ls -lh "$fichier" | awk '{print $5}')
23     echo -e "${CYAN}📏 Taille : $taille${NC}"
24
25     # Permissions
26     permissions=$(ls -l "$fichier" | awk '{print $1}')
27     echo -e "${CYAN}🔒 Permissions : $permissions${NC}"
28
29     # Date de modification
30     date_modif=$(ls -l "$fichier" | awk '{print $6, $7, $8}')
31     echo -e "${CYAN}📅 Dernière modification : $date_modif${NC}"
32
```

# Solution Exercice 2.4

```
● ● ●

1 # Contenu pour les fichiers texte
2     case "$nom_fichier" in
3         *.txt|*.md|*.sh|*.py|*.js|*.html|*.css)
4             if [ -r "$fichier" ]; then
5                 lignes=$(wc -l < "$fichier" 2>/dev/null || echo "?")
6                 echo -e "${CYAN} ${ICONES[1]} Nombre de lignes : $lignes${NC}"
7             fi
8             ;;
9     esac
10
11    # Vérification si exécutable
12    if [ -x "$fichier" ]; then
13        echo -e "${GREEN} ${ICONES[2]} Fichier exécutable${NC}"
14    fi
15
16    elif [ -d "$fichier" ]; then
17        echo -e "${BLUE} ${ICONES[3]} C'est un répertoire${NC}"
18        nb_elements=$(ls -1 "$fichier" 2>/dev/null | wc -l)
19        echo -e "${CYAN} ${ICONES[1]} Contient : $nb_elements éléments${NC}"
20    fi
21
22    # Résumé coloré
23    echo -e "${YELLOW} ${ICONES[2]} Résumé : $(basename "$fichier") = $([ -f "$f
24 }
```

# Solution Exercice 2.4

```
1
2 # Traitement de tous les fichiers passés en paramètre
3 echo -e "${BLUE}🔍 CLASSIFICATEUR DE FICHIERS${NC}"
4 echo -e "${BLUE}===== ${NC}"
5
6 total_fichiers=$#
7 fichiers_traites=0
8 fichiers_inexistants=0
9
10 for fichier in "$@"; do
11     if classifier_fichier "$fichier"; then
12         ((fichiers_traites++))
13     else
14         ((fichiers_inexistants++))
15     fi
16 done
17
18 # Statistiques finales
19 echo -e "\n${BLUE}== STATISTIQUES == ${NC}"
20 echo -e "${GREEN}✓ Fichiers analysés : $fichiers_traites/$total_fichiers"
21 if [ $fichiers_inexistants -gt 0 ]; then
22     echo -e "${RED}✗ Fichiers inexistants : $fichiers_inexistants${NC}"
23 fi
```

# Pause

 15 minutes

**Après la pause :**

- Boucles (for, while, until)
- Contrôle de boucles
- Fonctions

# **Séance 2 - Après-midi**

## **Boucles et Fonctions**

**(3h30)**

# **Boucles**

## **(2h30)**

# Boucle for - Liste



```
1 #!/bin/bash
2
3 # Parcours d'une liste
4 for fruit in pomme banane orange; do
5     echo "Fruit : $fruit"
6 done
7
8 # Parcours d'un tableau
9 fruits=("pomme" "banane" "orange" "kiwi")
10 for fruit in "${fruits[@]}"; do
11     echo "Fruit du tableau : $fruit"
12 done
13
14 # Parcours des fichiers
15 for fichier in *.txt; do
16     echo "Fichier texte trouvé : $fichier"
17 done
```

# Boucle for - Plage Numérique



```
1 #!/bin/bash
2
3 # Plage simple
4 for i in {1..5}; do
5     echo "Nombre : $i"
6 done
7
8 # Plage avec pas
9 for i in {0..20..2}; do # De 0 à 20, par pas de 2
10    echo "Nombre pair : $i"
11 done
12
13 # Plage avec variables
14 debut=1
15 fin=10
16 for i in $(seq $debut $fin); do
17     echo "Séquence : $i"
18 done
```

# Boucle for - Style C



```
1 #!/bin/bash
2
3 # Boucle style C
4 for ((i=1; i<=10; i++)); do
5     echo "Itération $i"
6 done
7
8 # Avec plusieurs variables
9 for ((i=1, j=10; i<=5; i++, j--)); do
10    echo "i=$i, j=$j"
11 done
12
13 # Parcours d'un tableau avec indices
14 tableau=("a" "b" "c" "d")
15 for ((i=0; i<${#tableau[@]}; i++)); do
16    echo "Index $i : ${tableau[i]}"
17 done
```

# Boucle while



```
1 #!/bin/bash
2
3 # Boucle while simple
4 compteur=1
5 while [ $compteur -le 5 ]; do
6     echo "Compteur : $compteur"
7     ((compteur++))
8 done
9
10 # Lecture d'un fichier ligne par ligne
11 while IFS= read -r ligne; do
12     echo "Ligne lue : $ligne"
13 done < "fichier.txt"
14
15 # Boucle infinie (attention !)
16 while true; do
17     echo "Boucle infinie - Ctrl+C pour arrêter"
18     sleep 1
19 done
```

# Boucle until

```
1 #!/bin/bash
2
3 # Boucle until (inverse de while)
4 compteur=1
5 until [ $compteur -gt 5 ]; do
6     echo "Compteur until : $compteur"
7     ((compteur++))
8 done
9
10 # Attendre qu'un fichier existe
11 until [ -f "signal.txt" ]; do
12     echo "Attente du fichier signal.txt..."
13     sleep 2
14 done
15 echo "Fichier trouvé !"
```

# Contrôle de Boucles

```
1 #!/bin/bash
2
3 # break : sortir de la boucle
4 for i in {1..10}; do
5     if [ $i -eq 5 ]; then
6         echo "Arrêt à $i"
7         break
8     fi
9     echo "Valeur : $i"
10 done
11
12 # continue : passer à l'itération suivante
13 for i in {1..10}; do
14     if [ $((i % 2)) -eq 0 ]; then
15         continue # Ignorer les nombres pairs
16     fi
17     echo "Nombre impair : $i"
18 done
```

# Boucles Imbriquées



```
1 #!/bin/bash
2
3 # Table de multiplication
4 echo "==== TABLE DE MULTIPLICATION ===="
5 for i in {1..5}; do
6     echo "Table de $i :"
7     for j in {1..5}; do
8         resultat=$((i * j))
9         printf "%d x %d = %2d    $i $j $resultat
10    done
11    echo # Nouvelle ligne
12 done
13
14 # Parcours d'une matrice
15 matrice=(
16     "1 2 3"
17     "4 5 6"
18     "7 8 9"
19 )
20
21 for ligne in "${matrice[@]}"; do
22     for valeur in $ligne; do
23         printf "%3d " $valeur
24     done
25     echo
26 done
```



# TP Pratique

## Traitement de Fichiers en Lot

**Durée :** 1h

# **Exercice 2.5 : Générateur de Fichiers de Test**

**Objectif :** Utiliser les boucles for

**À créer :** generateur.sh

**Fonctionnalités :**

- Créer 10 fichiers "test\_01.txt" à "test\_10.txt"
- Écrire dans chaque fichier : "Fichier de test numéro X"
- Créer 5 répertoires "dossier\_A" à "dossier\_E"
- Placer 2 fichiers dans chaque répertoire
- Afficher un résumé des créations

# Solution Exercice 2.5



```
1 #!/bin/bash
2 # generateur.sh - Générateur de fichiers de test
3
4 # Couleurs
5 RED='\033[0;31m'
6 GREEN='\033[0;32m'
7 BLUE='\033[0;34m'
8 YELLOW='\033[1;33m'
9 CYAN='\033[0;36m'
10 NC='\033[0m'
11
12 # variables de configuration
13 REPERTOIRE_BASE="fichiers_test"
14 NB_FICHIERS_RACINE=10
15 NB_DOSSIERS=5
16 NB_FICHIERS_PAR_DOSSIER=2
17
18 echo -e "${BLUE}█ GÉNÉRATEUR DE FICHIERS DE TEST${NC}"
19 echo -e "${BLUE}===== ${NC}"
20
```

# Solution Exercice 2.5



```
1 # Création du répertoire de base
2 if [ -d "$REPERTOIRE_BASE" ]; then
3     echo -e "${YELLOW}⚠️ Le répertoire $REPERTOIRE_BASE existe déjà${NC}"
4     read -p "Voulez-vous le supprimer et recommencer ? (o/N) : " choix
5     if [[ $choix =~ ^[Oo]$ ]]; then
6         rm -rf "$REPERTOIRE_BASE"
7         echo -e "${RED}🗂 Répertoire supprimé${NC}"
8     else
9         echo -e "${RED}❌ Arrêt du script${NC}"
10        exit 1
11    fi
12 fi
13
14 mkdir -p "$REPERTOIRE_BASE"
15 echo -e "${GREEN}✓ Répertoire de base créé : $REPERTOIRE_BASE${NC}"
16
17 # Compteurs pour statistiques
18 total_fichiers_crees=0
19 total_dossiers_crees=0
20
21 # 1. Création des fichiers dans le répertoire racine
22 echo -e "\n${CYAN}📄 Création des fichiers racine...${NC}"
23 for i in $(seq -w 1 $NB_FICHIERS_RACINE); do
24     fichier="$REPERTOIRE_BASE/test_$i.txt"
25
26     # Contenu du fichier avec métadonnées
27     cat > "$fichier" << EOF
```

# Solution Exercice 2.5



```
1 Fichier de test numéro $i
2 =====
3
4 Informations :
5 - Nom du fichier : $(basename "$fichier")
6 - Date de création : $(date '+%Y-%m-%d %H:%M:%S')
7 - Numéro : $i
8 - Taille approximative : Petit fichier de test
9 - Générateur : $0
10
11 Contenu aléatoire pour test :
12 $(head -c 50 /dev/urandom | base64 | head -1)
13
14 Fin du fichier test $i
15 EOF
16
17     echo -e "${GREEN} ✓ Crée : $(basename "$fichier")${NC}"
18     ((total_fichiers_crees++))
19 done
20
```

# Solution Exercice 2.5



```
1 # 2. Création des répertoires avec lettres A-E
2 echo -e "\n${CYAN}📁 Création des sous-répertoires...${NC}"
3 for i in $(seq 1 $NB_DOSSIERS); do
4     # Conversion en lettre (A=1, B=2, etc.)
5     lettre=$(printf "\\$(printf '%03o' $((64+i)))")
6     dossier="$REPERTOIRE_BASE/dossier_${lettre}"
7
8     mkdir -p "$dossier"
9     echo -e "${GREEN} ✓ Répertoire créé : $(basename \"$dossier\")${NC}"
10    ((total_dossiers_crees++))
11
12    # Création des fichiers dans chaque répertoire
13    echo -e "${YELLOW}    📄 Ajout de fichiers dans dossier_${lettre}...${NC}"
14    for j in $(seq 1 $NB_FICHIERS_PAR_DOSSIER); do
15        fichier_dossier="$dossier/fichier_${lettre}_${j}.txt"
16
17        cat > "$fichier_dossier" << EOF
18 Fichier $j du dossier ${lettre}
19 =====
20
21 Emplacement : $dossier
22 Fichier numéro : $j
23 Dossier : ${lettre}
24 Date : $(date '+%Y-%m-%d %H:%M:%S')
25 Taille du répertoire parent : $(du -sh "$REPERTOIRE_BASE" | cut -f1)
26
27 Test de contenu pour le fichier $j du dossier ${lettre}.
28 Ce fichier fait partie d'un ensemble de test généré automatiquement.
29
30
```

# Solution Exercice 2.5



```
1 Données de test :
2 $(date | md5sum | cut -d' ' -f1)
3
4 Fin du fichier.
5 EOF
6
7         echo -e "${GREEN}    ✓ Crée : $(basename "$fichier_dossier")${NC}"
8         ((total_fichiers_crees++))
9     done
10 done
11
12 # 3. Création d'un fichier index
13 echo -e "\n${CYAN}█ Création du fichier index...${NC}"
14 index_file="$REPERTOIRE_BASE/INDEX.txt"
15 cat > "$index_file" << EOF
16 INDEX DES FICHIERS GÉNÉRÉS
17 =====
18
19 Date de génération : $(date '+%Y-%m-%d %H:%M:%S')
20 Script générateur : $0
21 Utilisateur : $(whoami)
22 Système : $(uname -s)
23
24 STRUCTURE CRÉÉE :
25 =====
26
27 Fichiers racine ($NB_FICHIERS_RACINE fichiers) :
28 EOF
29
```

# Solution Exercice 2.5



```
1 # Ajout de la liste des fichiers racine
2 for i in $(seq -w 1 $NB_FICHIERS_RACINE); do
3     echo " - test_$i.txt" >> "$index_file"
4 done
5
6 echo "" >> "$index_file"
7 echo "Répertoires ($NB_DOSSIERS dossiers) :" >> "$index_file"
8
9 # Ajout de la liste des répertoires et leurs contenus
10 for i in $(seq 1 $NB_DOSSIERS); do
11     lettre=$(printf "\\$(printf '%03o' $((64+i)))")
12     echo " - dossier_${lettre}/" >> "$index_file"
13     for j in $(seq 1 $NB_FICHIERS_PAR_DOSSIER); do
14         echo "     * fichier_{${lettre}}_${j}.txt" >> "$index_file"
15     done
16 done
17
```

# Solution Exercice 2.5

```
1 # Ajout des statistiques
2 cat >> "$index_file" << EOF
3
4 STATISTIQUES :
5 =====
6 - Total fichiers : $total_fichiers_crees
7 - Total dossiers : $total_dossiers_crees
8 - Taille totale : $(du -sh "$REPERTOIRE_BASE" | cut -f1)
9
10 COMMANDES UTILES :
11 =====
12 - Lister tout : find $REPERTOIRE_BASE -type f
13 - Compter fichiers : find $REPERTOIRE_BASE -type f | wc -l
14 - Taille détaillée : du -ah $REPERTOIRE_BASE
15 - Nettoyer : rm -rf $REPERTOIRE_BASE
16 EOF
17
18 echo -e "${GREEN}✓ Index créé : $(basename "$index_file")${NC}"
19
20 # 4. Résumé final avec statistiques
21 echo -e "\n${BLUE}[RÉSUMÉ DE GÉNÉRATION${NC}"
22 echo -e "${BLUE}===== ${NC}"
23 echo -e "${GREEN}✓ Fichiers créés : $total_fichiers_crees${NC}"
24 echo -e "${GREEN}✓ Dossiers créés : $total_dossiers_crees${NC}"
25 echo -e "${GREEN}✓ Taille totale : $(du -sh "$REPERTOIRE_BASE" | cut -f1
26
27 # Arborescence visuelle
28 echo -e "\n${CYAN} ARBORESCENCE CRÉÉE : ${NC}"
29 if command -v tree >/dev/null 2>&1; then
30     tree "$REPERTOIRE_BASE"
31 else
```

# Solution Exercice 2.5



```
1
2 # Conseils d'utilisation
3 echo -e "\n${YELLOW}💡 CONSEILS D'UTILISATION :${NC}"
4 echo -e "${YELLOW}• Voir l'index complet : cat $REPERTOIRE_BASE/INDEX.txt$"
5 echo -e "${YELLOW}• Lister tous les fichiers : find $REPERTOIRE_BASE -type"
6 echo -e "${YELLOW}• Rechercher dans les fichiers : grep -r \"test\" $REPER"
7 echo -e "${YELLOW}• Nettoyer : rm -rf $REPERTOIRE_BASE${NC}"
8
9 echo -e "\n${GREEN}🎉 Génération terminée avec succès !${NC}"
```

# Exercice 2.6 : Moniteur de Services

**Objectif :** Utiliser while pour la surveillance

**À créer :** moniteur.sh

**Fonctionnalités :**

- Surveiller l'existence d'un fichier "stop.txt"
- Tant que le fichier n'existe pas :
  - Afficher l'heure et "Service en cours..."
  - Attendre 5 secondes
  - Vérifier la charge système
- Quand "stop.txt" existe, afficher "Arrêt demandé"

# Solution Exercice 2.6



```
1 #!/bin/bash
2 # moniteur.sh - Moniteur de services
3
4 # Couleurs
5 RED='\033[0;31m'
6 GREEN='\033[0;32m'
7 BLUE='\033[0;34m'
8 YELLOW='\033[1;33m'
9 CYAN='\033[0;36m'
10 PURPLE='\033[0;35m'
11 NC='\033[0m'
12
13 # Configuration
14 FICHIER_STOP="stop.txt"
15 FICHIER_LOG="moniteur.log"
16 INTERVALLE=5
17 MAX_ITERATIONS=20 # Limite de sécurité
18 SEUIL_CHARGE=2.0 # Seuil d'alerte pour la charge
19
20 # Variables
21 iteration=0
22 debut=$(date +%s)
23
24 # Fonction de log
25 log_message() {
26     local message="$1"
27     local timestamp=$(date '+%Y-%m-%d %H:%M:%S')
28     echo "[${timestamp}] ${message}" >> "$FICHIER_LOG"
29 }
30
31 # Fonction d'affichage avec couleurs
```

# Fonctions Avancées



```
1 #!/bin/bash
2
3 # Fonction avec nombre variable d'arguments
4 calculer_somme() {
5     local somme=0
6     local nb_args=$#
7
8     echo "Calcul de la somme de $nb_args nombres :"
9
10    for nombre in "$@"; do
11        echo "    + $nombre"
12        somme=$((somme + nombre))
13    done
14
15    echo "= $somme"
16    return $somme
17 }
18
19 # Fonction récursive (factorielle)
20 factorielle() {
21     local n=$1
22
23     if [ $n -le 1 ]; then
24         echo 1
25     else
26         local prev=$(factorielle $((n - 1)))
27         echo $((n * prev))
28     fi
29 }
30
31 # Tests
```



# TP Pratique

## Bibliothèque de Fonctions Utilitaires

Durée : 1h

# Exercice 2.7 : Bibliothèque Mathématique

**Objectif :** Créer et utiliser des fonctions

**À créer :** math\_lib.sh

**Fonctions à implémenter :**

- addition(a, b) - retourne la somme
- factorielle(n) - calcule n!
- est\_pair(n) - retourne 0 si pair, 1 si impair
- pgcd(a, b) - plus grand commun diviseur

**Bonus :** Validation des paramètres

# Solution Exercice 2.7



```
1 #!/bin/bash
2 # math_lib.sh - Bibliothèque mathématique
3
4 # Couleurs pour l'affichage
5 RED='\033[0;31m'
6 GREEN='\033[0;32m'
7 BLUE='\033[0;34m'
8 YELLOW='\033[1;33m'
9 CYAN='\033[0;36m'
10 NC='\033[0m'
11
12 # =====
13 # FONCTIONS UTILITAIRES
14 # =====
15
16 # Validation qu'un paramètre est un nombre entier
17 est_nombre_entier() {
18     local valeur=$1
19
20     if [[ $valeur =~ ^-[0-9]+$ ]]; then
21         return 0 # C'est un nombre entier
22     else
23         return 1 # Ce n'est pas un nombre entier
24     fi
25 }
26
27 # Affichage d'erreur standardisé
28 erreur() {
29     echo -e "${RED}✖ Erreur : ${1}${NC}" >&2
30 }
31
```

# Exercice 2.8 : Utilitaires Fichiers

**Objectif :** Fonctions pour manipulation de fichiers

**À créer :** fichier\_utils.sh

**Fonctions à implémenter :**

- `backup_file(fichier)` - crée une copie avec timestamp
- `compte_lignes(fichier)` - retourne le nombre de lignes
- `nettoie_log(fichier, jours)` - supprime les entrées anciennes
- `rapport_taille(repertoire)` - taille totale des fichiers

**Bonus :** Gestion d'erreurs dans chaque fonction

# Solution Exercice 2.8



```
1 #!/bin/bash
2 # fichier_utils.sh - Utilitaires de gestion de fichiers
3
4 # Couleurs
5 RED='\033[0;31m'
6 GREEN='\033[0;32m'
7 BLUE='\033[0;34m'
8 YELLOW='\033[1;33m'
9 CYAN='\033[0;36m'
10 PURPLE='\033[0;35m'
11 NC='\033[0m'
12
13 # =====
14 # FONCTIONS UTILITAIRES
15 # =====
16
17 # Affichage d'erreur
18 erreur() {
19     echo -e "${RED}✖ Erreur : $1${NC}" >&2
20 }
21
22 # Affichage de succès
23 succès() {
24     echo -e "${GREEN}✓ $1${NC}"
25 }
26
27 # Affichage d'information
28 info() {
29     echo -e "${CYAN}ℹ $1${NC}"
30 }
31
```

# Récapitulatif Séance 2 Complète

## Matinée :

- Conditions et tests (if/then/else, opérateurs)
- Structures case/esac avec patterns
- Tests avancés et expressions régulières

## Après-midi :

- Boucles (for, while, until)
- Contrôle de boucles (break, continue)
- Fonctions avec paramètres et valeurs de retour

# Compétences Acquises

- 🎯 Maîtriser les structures conditionnelles
- 🎯 Implémenter des boucles efficaces
- 🎯 Créer et utiliser des fonctions
- 🎯 Développer des scripts modulaires
- 🎯 Gérer les flux de contrôle complexes

# **Préparation Séance 3**

**Prochaine séance : Manipulation de fichiers et texte**

**À réviser :**

- Conditions et boucles
- Fonctions créées aujourd'hui
- Tests sur fichiers

**À préparer :**

- Scripts de la séance 2 sauvegardés
- Environnement avec fichiers de test

# Questions Finales ?

Excellente progression ! 

# **Formation Bash Scripting - Session 3 Manipulation de fichiers et texte**

# **Session 3 (7h)**

## **Manipulation de fichiers et texte**

**Objectifs :**

- Maîtriser les redirections et pipes
- Utiliser grep et les expressions régulières
- Manipuler les fichiers avec sed
- Traiter les données avec awk

# Plan de la session

## Matin (3h30)

- **1h30** - Redirection et pipes
- **2h** - Expressions régulières et grep

## Après-midi (3h30)

- **2h** - sed - Éditeur de flux
- **1h30** - awk - Traitement de données

# Partie 1 : Redirections et pipes

## 1h30

# Redirections de base

## Redirections des flux

```
1 # Redirection de la sortie standard
2 ls > fichiers.txt
3
4 # Redirection en mode ajout
5 echo "nouvelle ligne" >> fichiers.txt
6
7 # Redirection de l'entrée standard
8 sort < fichiers.txt
9
10 # Redirection des erreurs
11 ls /inexistant 2> erreurs.log
12
13 # Redirection complète (stdout + stderr)
14 ls /inexistant &> tout.log
```

# Pipes - Chaînage de commandes

## Concept fondamental

```
1 # Pipe simple
2 ls -l | grep ".txt"
3
4 # Chaînage multiple
5 ps aux | grep apache | wc -l
6
7 # Combinaison avec redirections
8 cat access.log | grep "404" | wc -l > erreurs_count.txt
```

**Principe :** La sortie d'une commande devient l'entrée de la suivante

# Here Documents et Here Strings

## Here document (<<)

```
● ● ●  
1 cat << EOF > config.txt  
2 ServerName monserveur.com  
3 DocumentRoot /var/www/html  
4 Listen 80  
5 EOF
```

## Here string (<<<)

```
● ● ●  
1 grep "pattern" <<< "chaîne à analyser"  
2  
3 # Équivalent à :  
4 echo "chaîne à analyser" | grep "pattern"
```

# Here Documents et Here Strings

<https://www.baeldung.com/linux/heredoc-herestring>

# TP Pratique - Exercice 3.1

## Analyseur de logs système

**Objectif :** Maîtriser redirections et pipes

**Tâches :**

- Lire un fichier de log (créer un fichier de test)
- Compter les erreurs avec grep et wc
- Extraire les 10 dernières connexions
- Sauvegarder le rapport avec timestamp
- Rediriger les erreurs vers un fichier séparé

# TP Pratique - Exercice 3.2

## Générateur de configuration

**Objectif :** Utiliser here documents

**Tâches :**

- Créer un générateur de configuration Apache
- Demander : nom du site, port, répertoire
- Utiliser here document pour le template
- Remplacer les variables dans le template
- Sauvegarder la configuration générée

# **Partie 2 : Expressions régulières et grep**

**2h**

# Expressions régulières - Syntaxe de base Métacaractères principaux

```
1 .          # N'importe quel caractère
2 ^          # Début de ligne
3 $          # Fin de ligne
4 *          # 0 ou plusieurs occurrences
5 +          # 1 ou plusieurs occurrences (egrep)
6 ?          # 0 ou 1 occurrence (egrep)
7 [ ]        # Classe de caractères
8 [ ^ ]      # Négation de classe
9 \          # Échappement
```

# Classes de caractères

## Classes prédéfinies



```
1 [0-9]          # Chiffres
2 [a-z]          # Lettres minuscules
3 [A-Z]          # Lettres majuscules
4 [a-zA-Z]        # Toutes les lettres
5 [:digit:]      # Chiffres (POSIX)
6 [:alpha:]       # Lettres (POSIX)
7 [:alnum:]       # Lettres et chiffres
8 [:space:]      # Espaces
```

# Commande grep et ses options

## Options principales



```
1 grep -i pattern file      # Insensible à la casse
2 grep -v pattern file      # Lignes ne contenant PAS le pattern
3 grep -n pattern file      # Numéros de lignes
4 grep -r pattern dir/       # Recherche récursive
5 grep -c pattern file      # Compte les occurrences
6 grep -l pattern files      # Noms des fichiers contenant le pattern
7 grep -E pattern file      # Extended regex (= egrep)
8 grep -F pattern file      # Fixed strings (= fgrep)
```

# Exemples pratiques avec grep

## Recherches courantes



```
1 # Adresses IP
2 grep -E "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}" access.log
3
4 # Adresses email
5 grep -E "[a-zA-Z0-9]+@[a-zA-Z0-9]+\.[a-zA-Z]{2,}" contacts.txt
6
7 # Lignes vides
8 grep "^$" fichier.txt
9
10 # Lignes non vides
11 grep -v "^$" fichier.txt
12
13 # Mots complets uniquement
14 grep -w "error" log.txt
```

# TP Pratique - Exercice

## 3.3

### Auditeur de configurations

**Objectif :** Utiliser grep et regex pour l'audit

**Tâches :**

- Rechercher les adresses IP dans les fichiers .conf
- Trouver les ports ouverts (format :port)
- Identifier les mots de passe en dur
- Lister les fichiers de log configurés
- Générer un rapport d'audit sécurisé

# TP Pratique - Exercice

## 3.4

### Extracteur d'emails et URLs

**Objectif :** Regex complexes avec grep

**Tâches :**

- Extraire toutes les adresses email valides
- Extraire toutes les URLs (http/https)
- Extraire les numéros de téléphone français
- Trier et dédupliquer les résultats
- Sauvegarder dans des fichiers séparés

# **Partie 3 : sed - Éditeur de flux**

**2h**

# sed - Principe de fonctionnement

## Stream Editor

- **Flux** : Traite les données ligne par ligne
- **Non-interactif** : Pas d'interface utilisateur
- **Puissant** : Remplacements, suppressions, insertions
- **Scriptable** : Intégrable dans les scripts



```
1 # Syntaxe générale
2 sed 'commande' fichier
3 sed -e 'commande1' -e 'commande2' fichier
4 sed -f script.sed fichier
```

# Commandes de base de sed

## Commandes principales



```
1 s/ancien/nouveau/      # Substitution
2 d                      # Suppression
3 p                      # Impression
4 a\texte                 # Ajout après la ligne
5 i\texte                 # Insertion avant la ligne
6 c\texte                 # Changement de ligne
7 =                      # Numéro de ligne
```

# Substitution avec sed

## Syntaxe et options

```
1 # Substitution simple (première occurrence)
2 sed 's/ancien/nouveau/' fichier
3
4 # Substitution globale (toutes les occurrences)
5 sed 's/ancien/nouveau/g' fichier
6
7 # Substitution insensible à la casse
8 sed 's/ancien/nouveau/gi' fichier
9
10 # Modification en place (attention !)
11 sed -i 's/ancien/nouveau/g' fichier
12
13 # Sauvegarde automatique
14 sed -i.bak 's/ancien/nouveau/g' fichier
```

# Adressage dans sed

## Spécifier les lignes à traiter

```
1 # Ligne spécifique
2 sed '5s/ancien/nouveau/' fichier
3
4 # Plage de lignes
5 sed '1,10s/ancien/nouveau/' fichier
6
7 # Depuis une ligne jusqu'à la fin
8 sed '5,$s/ancien/nouveau/' fichier
9
10 # Pattern d'adressage
11 sed '/pattern/s/ancien/nouveau/' fichier
12
13 # Négation
14 sed '/pattern/!s/ancien/nouveau/' fichier
```

# Exemples avancés avec sed

## Cas d'usage courants



```
1 # Supprimer les lignes vides
2 sed '/^$/d' fichier
3
4 # Supprimer les commentaires
5 sed '/^#/d' fichier
6
7 # Ajouter du texte au début
8 sed '1i\# Configuration générée automatiquement' config
9
10 # Remplacer avec références
11 sed 's/\\([0-9]*\\)\\.\\([0-9]*\\)/Version \\1 build \\2/' versions
12
13 # Plusieurs commandes
14 sed -e 's/foo/bar/g' -e '/^$/d' fichier
```

# TP Pratique - Exercice 3.5

## Configurateur automatique SSH

**Objectif :** Modifier des fichiers de configuration avec sed

### Tâches :

- Modifier /etc/ssh/sshd\_config (ou copie de test)
- Changer le port par défaut (22 → 2222)
- Désactiver l'authentification par mot de passe root
- Activer les logs détaillés
- Créer une sauvegarde avant modification

# TP Pratique - Exercice 3.6

## Nettoyeur de fichiers de log

**Objectif :** Traitement de données avec sed

**Tâches :**

- Supprimer les lignes vides et commentaires
- Anonymiser les adresses IP (XXX.XXX.XXX.XXX)
- Supprimer les timestamps anciens (> 30 jours)
- Normaliser le format des dates
- Remplacer les mots de passe par [MASKED]

# Partie 4 : awk - Traitement de données

1h30

# awk - Introduction

## Langage de traitement de données

- **Orienté colonnes** : Traite les données structurées
- **Pattern-Action** : Condition → Action
- **Variables intégrées** : NR, NF, \$0, \$1, \$2...
- **Fonctions** : Mathématiques, chaînes, E/S



```
1 # Syntaxe générale
2 awk 'pattern { action }' fichier
3 awk -F'delimiter' 'program' fichier
```

# Variables prédéfinies awk

## Variables importantes



```
1 NR                  # Numéro de ligne (Number of Record)
2 NF                  # Nombre de champs (Number of Fields)
3 $0                  # Ligne complète
4 $1, $2...           # Premier champ, deuxième champ...
5 FS                  # Séparateur de champs (Field Separator)
6 OFS                 # Séparateur de sortie (Output Field Separator)
7 RS                  # Séparateur d'enregistrements
8 ORS                 # Séparateur de sortie d'enregistrements
```

# Structure d'un programme awk

## Pattern { Action }



```
1 # Action sur toutes les lignes
2 awk '{ print $1 }' fichier
3
4 # Pattern simple
5 awk '/pattern/ { print }' fichier
6
7 # Pattern avec condition
8 awk '$3 > 100 { print $1, $3 }' fichier
9
10 # Blocs BEGIN et END
11 awk 'BEGIN { print "Début" } { print } END { print "Fin" }' fichier
```

# Exemples pratiques avec awk

## Cas d'usage courants



```
1 # Afficher une colonne spécifique
2 awk '{ print $2 }' fichier
3
4 # Somme d'une colonne
5 awk '{ sum += $3 } END { print sum }' fichier
6
7 # Moyenne d'une colonne
8 awk '{ sum += $3; n++ } END { print sum/n }' fichier
9
10 # Compter les lignes par pattern
11 awk '/ERROR/ { count++ } END { print count }' log
12
13 # Formatage personnalisé
14 awk '{ printf "%-10s %5d\n", $1, $2 }' fichier
```

# Traitement de fichiers CSV avec awk

## Données structurées



```
1 # Changer le séparateur de champs
2 awk -F',' '{ print $1, $3 }' data.csv
3
4 # Calculer sur des colonnes numériques
5 awk -F',' '$4 > 1000 { print $1, $4 }' sales.csv
6
7 # Grouper par catégorie
8 awk -F',' '{ total[$2] += $3 } END { for (cat in total) print cat, total[cat] }'
```

# TP Pratique - Exercice

## 3.7

### Analyseur de logs Apache

**Objectif :** Extraire des statistiques avec awk

#### Tâches :

- Compter le nombre total de requêtes
- Lister les 10 IP les plus fréquentes
- Calculer la bande passante totale
- Identifier les erreurs 404 et 500
- Générer un rapport formaté

**Format log :** IP - - [date] "méthode URL" code taille

# TP Pratique - Exercice

## 3.8

### Processeur de fichier CSV

**Objectif :** Manipuler des données structurées

**Tâches :**

- Calculer le salaire moyen par département
- Trouver l'employé le mieux payé
- Lister les employés embauchés cette année
- Générer un rapport formaté en colonnes
- Exporter les résultats en CSV

**Format :**

nom,prenom,departement,salaire,date\_embauche

# Fonctions avancées awk

## Fonctions intégrées utiles



```
1 # Fonctions de chaînes
2 length(string)          # Longueur
3 substr(string, start, length) # Sous-chaîne
4 gsub(/pattern/, replacement, string) # Remplacement global
5 split(string, array, separator) # Division en tableau
6
7 # Fonctions mathématiques
8 int(x)                  # Partie entière
9 sqrt(x)                 # Racine carrée
10 rand()                  # Nombre aléatoire
11
12 # Fonctions de formatage
13 printf(format, args)    # Sortie formatée
14 sprintf(format, args)   # Chaîne formatée
```

# Combinaisons puissantes

## Chainer les outils



```
1 # grep + sed + awk
2 cat access.log | \
3   grep "POST" | \
4   sed 's/POST/[POST]/' | \
5   awk '{ print $1, $7, $9 }'
6
7 # Traitement complexe en une ligne
8 awk '/ERROR/ { gsub(/[^0-9]{1,3}\.[^0-9]{1,3}\.[^0-9]{1,3}\.[^0-9]{1,3}/, "XXX"
```

# Bonnes pratiques

## Conseils d'utilisation

- **grep** : Recherche simple de motifs
- **sed** : Modifications de texte, substitutions
- **awk** : Traitement de données structurées, calculs
- **Combiner** : Utiliser les pipes pour chaîner
- **Tester** : Toujours tester sur un échantillon
- **Sauvegarder** : Faire une copie avant modification

# Récapitulatif Session 3

## Ce que nous avons appris

-  **Redirections et pipes** : Maîtrise des flux de données
-  **grep et regex** : Recherche et filtrage avancés
-  **sed** : Modification automatique de fichiers
-  **awk** : Traitement de données structurées

## Compétences acquises

- Automatiser le traitement de logs
- Manipuler des fichiers de configuration
- Extraire et analyser des données
- Créer des outils d'audit système

# **Questions / Discussion**

## **Prochaine session (Session 4)**

- Gestion des erreurs et débogage
- Tableaux et données structurées
- Processus et signaux
- Communication inter-processus

**Techniques avancées pour des scripts robustes !**

# Merci !

## Session 3 terminée

Manipulation de fichiers et texte

Pratiquez les exercices pour consolider vos acquis

## Projet Final - Système de Sauvegarde

## Automatisé

## Formation Bash Scripting - Séances 1 à 3



# Objectifs du Projet

Créer un système de sauvegarde automatisé professionnel en utilisant bash script.



# Description Générale

Vous devez développer un script `backup_system.sh` qui offre un système complet de sauvegarde avec :

- Interface utilisateur intuitive (menu interactif)
- Gestion de profils de sauvegarde
- Création d'archives tar.gz avec compression
- Logs détaillés et rapports
- Gestion d'erreurs robuste
- Fonctionnalités avancées (rotation, vérification, restauration)

# Structure du Projet - 4 Parties

## Partie 1 : Interface et Menu Principal

**Compétences mobilisées :** Variables, paramètres, conditions, menus (case/esac)

## Partie 2 : Système de Sauvegarde de Base

**Compétences mobilisées :** Fonctions, boucles, gestion de fichiers, redirections

## Partie 3 : Logs et Analyse des Données

**Compétences mobilisées :** grep, sed, awk, pipes, traitement de texte

## Partie 4 : Fonctionnalités Avancées et Finalisation

# PARTIE 1 : Interface et Menu Principal

*Durée estimée : 1h30*

## Objectifs de la Partie 1

Créer l'interface utilisateur et la structure de base du script.

### Fonctionnalités à implémenter :

#### 1.1 Structure de base du script



```
1 #!/bin/bash
2 # En-tête avec informations du script
3 # Variables globales
4 # Fonctions utilitaires de base
```

## **1.2 Menu principal interactif**

Créer un menu avec les options suivantes :

- 1. Créer une sauvegarde**
- 2. Lister les sauvegardes**
- 3. Restaurer une sauvegarde**
- 4. Voir les logs**
- 5. Configuration**
- 6. Aide**
- 7. Quitter**

## 1.3 Gestion des paramètres en ligne de commande

Le script doit accepter :

- `./backup_system.sh --help` : Afficher l'aide
- `./backup_system.sh --version` : Afficher la version
- `./backup_system.sh --config` : Aller directement à la configuration
- `./backup_system.sh --backup [profil]` : Lancer une sauvegarde directement

## 1.4 Configuration initiale

Demander à l'utilisateur :

- Répertoire de destination des sauvegardes
- Répertoire des logs
- Nom d'utilisateur/email pour les notifications
- Préférences de compression

# Critères d'évaluation Partie 1 :

-  Menu fonctionnel avec gestion des choix invalides
-  Paramètres en ligne de commande opérationnels
-  Variables globales bien organisées
-  Configuration sauvegardée dans un fichier
-  Interface utilisateur claire et intuitive

# PARTIE 2 : Système de Sauvegarde de Base

*Durée estimée : 2h*

## Objectifs de la Partie 2

Implémenter le cœur du système de sauvegarde.

Fonctionnalités à implémenter :

### 2.1 Fonction de sauvegarde principale



```
1 # Fonction create_backup()
2 # - Vérification de l'espace disque disponible
3 # - Création de l'archive tar.gz
4 # - Gestion de l'horodatage
5 # - Calcul de la taille et du temps d'exécution
```

## 2.2 Gestion des profils de sauvegarde

Créer un système de profils avec :

- **Profil par défaut** : Documents utilisateur
- **Profil système** : Fichiers de configuration
- **Profil personnalisé** : Répertoires au choix de l'utilisateur

## 2.3 Sélection et validation des répertoires source

- Interface pour choisir les répertoires à sauvegarder
- Validation de l'existence des répertoires
- Calcul de la taille totale à sauvegarder
- Gestion des exclusions (fichiers temporaires, caches)

## 2.4 Cration d'archives avec metadonnees

- Nom d'archive :  
`backup_[profil]_YYYYMMDD_HHMMSS.tar.gz`
- Fichier de metadonnees associe (.info)
- Verification de l'integrite de l'archive cree
- Calcul du checksum MD5/SHA256

## 2.5 Gestion des erreurs et interruptions

- Verification de l'espace disque avant sauvegarde
- Gestion de l'interruption (Ctrl+C) avec nettoyage
- Messages d'erreur explicites
- Rollback en cas d'chec

## Critères d'évaluation Partie 2 :

-  Archives tar.gz créées correctement
-  Système de profils fonctionnel
-  Gestion robuste des erreurs
-  Métadonnées complètes et exploitables
-  Performance et optimisation

# PARTIE 3 : Logs et Analyse des Données

*Durée estimée : 1h30*

## Objectifs de la Partie 3

Implémenter le système de logs et d'analyse des sauvegardes.

### Fonctionnalités à implémenter :

#### 3.1 Système de logging complet



```
1 # Niveaux de log : DEBUG, INFO, WARN, ERROR  
2 # Format : [TIMESTAMP] [LEVEL] [MODULE] Message  
3 # Rotation automatique des logs
```

## 3.2 Rapports de sauvegarde

Générer des rapports avec :

- Statistiques de la sauvegarde (taille, durée, fichiers)
- Liste des fichiers inclus/exclus
- Erreurs et avertissements rencontrés
- Comparaison avec les sauvegardes précédentes

## 3.3 Analyse des logs avec grep/sed/awk

- **Fonction d'analyse des erreurs** : Extraction des messages d'erreur
- **Statistiques d'utilisation** : Fréquence des sauvegardes, profils utilisés
- **Détection d'anomalies** : Tailles inhabituelles, durées anormales
- **Graphiques ASCII** : Évolution de la taille des sauvegardes

### **3.4 Recherche et filtrage des sauvegardes**

- Recherche par date, profil, taille
- Filtrage des logs par niveau de gravité
- Export des statistiques en CSV
- Nettoyage automatique des anciens logs

### **Critères d'évaluation Partie 3 :**

-  Logs structurés et exploitables
-  Rapports détaillés et informatifs
-  Utilisation efficace de grep/sed/awk
-  Statistiques pertinentes et visuelles
-  Fonctions de recherche opérationnelles

# PARTIE 4 : Fonctionnalités Avancées et Finalisation

*Durée estimée : 2h*

## Objectifs de la Partie 4

Compléter le système avec des fonctionnalités professionnelles.

### Fonctionnalités à implémenter :

#### 4.1 Système de restauration



```
1 # Fonction restore_backup()
2 # - Sélection de l'archive à restaurer
3 # - Prévisualisation du contenu
4 # - Restauration complète ou sélective
5 # - Vérification de l'intégrité avant restauration
```

## 4.2 Rotation automatique des sauvegardes

- Politique de rétention configurable
- Suppression automatique des anciennes sauvegardes
- Préservation des sauvegardes importantes
- Optimisation de l'espace disque

## 4.3 Notification et monitoring

- Notifications par email (simulation)
- Statut de santé du système de sauvegarde
- Alertes en cas de problème (espace disque, échecs)
- Interface de monitoring simple

## 4.4 Sécurisation et optimisation

- Chiffrement optionnel des archives
- Validation des entrées utilisateur
- Optimisation des performances (compression, parallélisation)
- Documentation intégrée et aide contextuelle

## 4.5 Tests et validation

- Jeu de tests automatisés
- Validation de tous les scénarios d'usage
- Gestion des cas limites
- Optimisation finale

## Critères d'évaluation Partie 4 :

-  Fonction de restauration complète
-  Rotation automatique opérationnelle
-  Système de notifications fonctionnel
-  Sécurité et robustesse
-  Documentation et tests complets



# Évaluation Globale du Projet

Critères d'évaluation (100 points) :

Critère	Points	Description
<b>Fonctionnalité</b>	40 pts	Toutes les fonctions demandées sont implémentées
<b>Code Quality</b>	20 pts	Structure, lisibilité, documentation
<b>Robustesse</b>	20 pts	Gestion d'erreurs, cas limites, sécurité
<b>Innovation</b>	10 pts	Fonctionnalités bonus créatives
<b>Présentation</b>	10 pts	Démonstration, explication du code

## Livrables attendus :

1. **Script principal** : backup\_system.sh
2. **Fichier de configuration** : backup\_config.conf
3. **Documentation** : README.md
4. **Jeu de tests** : test\_backup.sh
5. **Rapport de projet** : Description des choix techniques



# Conseils et Ressources

## Structure de fichiers recommandée :

```
1 projet_backup/
2   └── backup_system.sh      # Script principal
3   └── backup_config.conf    # Configuration
4   └── README.md             # Documentation
5   └── test_backup.sh        # Tests
6   └── logs/                 # Répertoire des logs
7   └── backups/              # Répertoire des sauvegardes
8   └── temp/                 # Fichiers temporaires
```

## Fonctions utiles à développer :

- `show_menu()` - Affichage du menu
- `create_backup()` - Création de sauvegarde
- `list_backups()` - Liste des sauvegardes
- `restore_backup()` - Restauration
- `log_message()` - Système de logging
- `check_disk_space()` - Vérification espace
- `cleanup()` - Nettoyage et rotation

## Bonnes pratiques :

- Utiliser `set -e` pour arrêter sur erreur
- Valider toutes les entrées utilisateur
- Créer des sauvegardes avant toute opération critique
- Documenter chaque fonction importante
- Tester régulièrement pendant le développement

JUL  
17

# Planning de Développement

## Séance de travail (3h30) :

- **30 min** : Lecture de l'énoncé et planification
- **1h30** : Partie 1 - Interface et menu
- **30 min** : Pause et validation Partie 1
- **2h** : Partie 2 - Système de sauvegarde
- **1h30** : Partie 3 - Logs et analyse
- **2h** : Partie 4 - Fonctionnalités avancées
- **30 min** : Tests finaux et documentation

## Présentation finale (15 min par groupe) :

1. **Démonstration** (8 min) : Présentation des fonctionnalités
2. **Code** (5 min) : Explication des parties techniques
3. **Questions** (2 min) : Réponses aux questions du formateur



# Critères de Réussite

## Niveau Minimum (Note : 10/20) :

- Menu principal fonctionnel
- Sauvegarde de base opérationnelle
- Logs simples
- Gestion d'erreurs de base

## Niveau Intermédiaire (Note : 14/20) :

- Toutes les parties 1-3 complètes
- Système de profils fonctionnel
- Analyse des logs avancée
- Interface utilisateur soignée

## Niveau Avancé (Note : 18+/20) :

- Projet complet avec toutes les parties
- Fonctionnalités bonus innovantes
- Code optimisé et sécurisé
- Documentation professionnelle

*Bonne chance pour ce projet final qui démontrera votre maîtrise du bash scripting !*