

# Introduction aux systèmes d'exploitation et réseaux

Philippe Rannou

ESIR

# Organisation du module

## Volume horaire

40 H (sur 15 semaines) :

- 8H de CM
- 32H de TP

## Intervenants

- CM : Philippe Rannou
- TP : Philippe Rannou et Romain Mounier

## Modalités d'évaluation


- TPs à rendre
- Projet

## Contenu

- Bases des systèmes d'exploitation
- Utilisation du terminal sous Linux
- Bases des réseaux

## Compétences visées

- Savoir utiliser les commandes classiques du terminal
- Concevoir des scripts bash
- Utiliser python pour dialoguer via le réseau

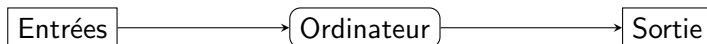
les exercices qui jalonnent ce cours sont signalés par un : , ils seront corrigés en cours et serviront en TDs/TPs.

## Plan du cours

- 1 Systèmes d'exploitation
- 2 Commandes du terminal
- 3 Scripts
- 4 Réseaux

## Qu'est-ce qu'un ordinateur ?

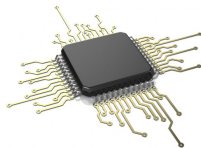
Un ordinateur est une machine électronique qui permet l'exécution de programmes.



# Composants d'un ordinateur

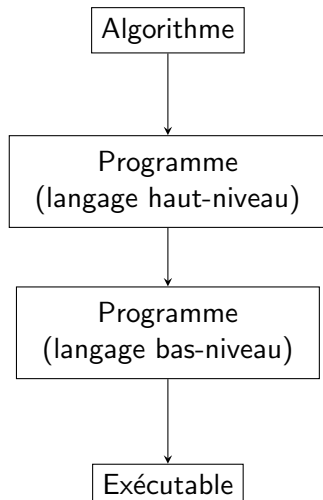
Un ordinateur est composé :

- d'un processeur : pour effectuer des calculs
- de mémoire : pour stocker les entrées et sorties de ces calculs
- de périphériques d'entrée/sortie : pour pouvoir saisir/récupérer les entrées et les sorties des programmes.



# Programme : de la conception à l'exécution

- Conception de l'algorithme
- Traduction de l'algorithme en langage de programmation (C, java, ...)
- Compilation en langage bas-niveau
- Génération d'un exécutable



# Répartition des disciplines informatiques

Bas niveau  
(proche machine)

Haut niveau  
(abstrait)

*Architecture  
matérielle*

*Système d'exploitation  
Réseau*

*Programmation  
python*

*Algorithmique*



- 1 Systèmes d'exploitation
  - GNU/Linux
- 2 Commandes du terminal
- 3 Scripts
- 4 Réseaux

# Qu'est-ce qu'un système d'exploitation ?

## Définition 1.1

*Un système d'exploitation (ou OS pour Operating System) est un ensemble de programmes qui dirige l'utilisation des ressources d'un ordinateur.*

*Source : Wikipedia*

## Rôle général

Le système d'exploitation a pour objectifs :

- simplifier l'accès aux ressources de l'ordinateur (mémoire/processeur/périphériques) pour les programmes,
- prévenir les interférences entre les programmes et les utilisateurs,
- de gérer ces ressources de manière efficace.

# Exemple 1

Les Entrées/Sorties disquettes du contrôleur NEC PD765 comprennent 16 commandes (Lecture, écriture, déplacement du bras, ...) :

- Les commandes `read` et `write` (par exemple) demandent 13 paramètres (compactés dans 9 octets) dont :
  - l'adresse du bloc à écrire,
  - le nombre de secteurs par piste,
  - l'espacement entre secteurs,
  - ...
- Le contrôleur retourne 23 champs de statuts et d'erreurs compactés dans 7 octets.
- Le programmeur doit également être conscient de l'activité du moteur : s'il est éteint, il doit l'activer avant la commande, et attendre une réponse avant d'écrire les données.

⇒ Le programmeur haut-niveau n'a envie que de commandes `read` et `write` prenant en paramètre le nom du disque.

## Exemple 2

Supposons 2 processus s'exécutant en parallèle sur un même ordinateur contenant 4 cases mémoire (numérotées 0, 1, 2 et 3).

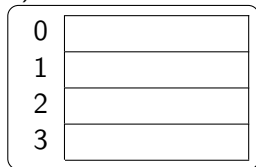
Processus 1 :

$a = 10$

$a = a + 3$

Processus 2 :

$b = 22$



Lors d'une session:

- le processus 1 effectue un calcul intermédiaire et inscrit son résultat 10 dans la case mémoire 2.
- le processus 2 effectue un autre calcul en parallèle et inscrit son résultat 22 dans la case mémoire 2.
- le processus 1 reprend le résultat de son calcul intermédiaire dans la case 2.

⇒ **Problème ! La case a été modifiée par le processus 2, le résultat final du processus 1 est corrompu !**

# Pourquoi est-ce important de s'intéresser aux systèmes d'exploitation ?

Le système d'exploitation propose une abstraction du matériel qui fait que l'utilisateur n'a *plus besoin* de s'en soucier !

⇒ **Démo**

Il existe de nombreux systèmes d'exploitation. Pour les ordinateurs, on peut les regrouper en 3 familles :

- Windows
- MacOS
- GNU/Linux (dont Ubuntu)

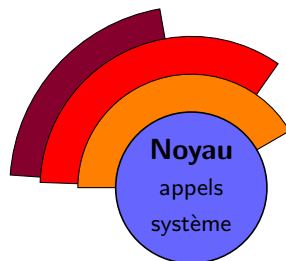
La suite de ce cours est basée sur le système **GNU/Linux**.

- 1 Systèmes d'exploitation
  - GNU/Linux
- 2 Commandes du terminal
- 3 Scripts
- 4 Réseaux

# Organisation du système d'exploitation GNU/Linux

Le système d'exploitation est organisé en couches :

- le cœur du système, appelé **Noyau** (*kernel*), responsable des appels systèmes au matériel.
- Les bibliothèques et outils systèmes (compilateur, etc.)
- Le terminal (*Shell*), qui fournit une CLI (*Command Line Interface*) pour dialoguer avec le noyau.
- La GUI (*Graphical User Interface*), qui permet à l'utilisateur d'interagir avec l'ordinateur de manière graphique.





## À l'origine

- a remplacé les cartes perforées
- unique moyen de communiquer avec l'ordinateur
- uniquement textuel



## Distinction Terminal/Shell/Console

- le **Terminal** et la **Console** représentaient l'appareil physique sur lequel on communiquait avec l'ordinateur.  
Actuellement, ils représentent un *émulateur* logiciel de cet appareil.
- Le **Shell** est le programme qui attend que l'utilisateur entre une commande.

## 1 Systèmes d'exploitation

## 2 Commandes du terminal

- Généralités sur le terminal
- Fichiers et dossiers
- Utilisateurs
- Entrées/Sorties et redirections
- Processus
- find et grep

## 3 Scripts

## 4 Réseaux

## Plusieurs shell existants

- sh, zsh, ksh, ...
- Sous Linux, nous utiliserons bash (*Bourne-Again SHell*)

## Langage de script

bash est également le nom du langage reconnu par le terminal.

### Caractéristiques :

- Pas de typage : tout est chaîne de caractère
- séparation par des espaces (pas de ' , ' ; ' ( ) ' etc.)
- *possibilité de faire des opérations arithmétiques avec des commandes spéciales.*

# Structure d'une commande

evince	--fullscreen	presentation.pdf
	'-----'	'-----'
v	v	v
nom	option(s)	argument(s)

En général :

- les options sont précédées de '-' ou '--'
- les arguments sont souvent des noms de fichiers et/ou dossiers

# Commandes liées au terminal 1

```
$ clear
```

→ Efface le terminal

```
$ history
```

→ Affiche toutes les commandes qui ont été entrées dans le terminal (également pour les sessions précédentes)

```
$ exit
```

→ (ou [Ctrl]+d) ferme le terminal

# Commandes liées au terminal 2

```
$ man commande
```

→ Affiche le manuel d'une commande. Navigation :

- ↑ et ↓ pour avancer/reculer d'une ligne,
- [espace] pour avancer d'une page, q pour quitter,
- /mot pour rechercher 'mot' dans le texte, puis n pour chercher le suivant ou [Shift]+n pour le précédent.

## Raccourci utiles

- [Tab] ×1 permet d'auto-compléter les noms de commande et les noms de fichier (si pas d'ambiguïté)
- [Tab] ×2 permet d'afficher les différentes possibilités
- [Ctrl]+c permet d'interrompre une commande (Attention !)
- [Ctrl]+[Shift]+c/[Ctrl]+[Shift]+v permet de copier/coller dans un terminal.

## 1 Systèmes d'exploitation

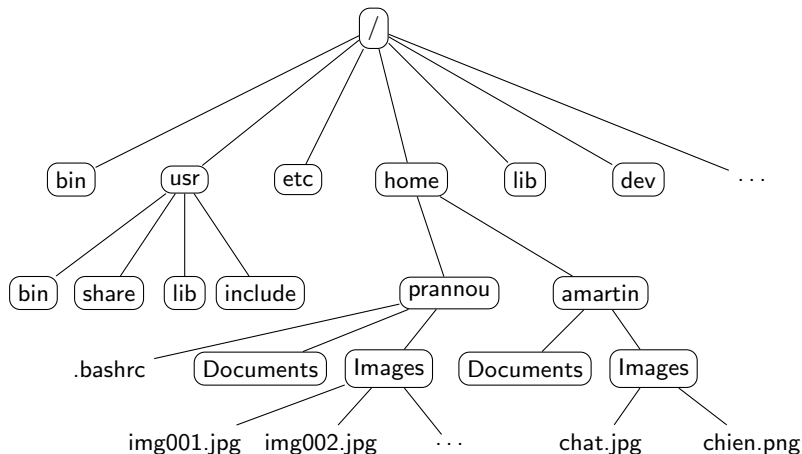
## 2 Commandes du terminal

- Généralités sur le terminal
- Fichiers et dossiers
- Utilisateurs
- Entrées/Sorties et redirections
- Processus
- find et grep

## 3 Scripts

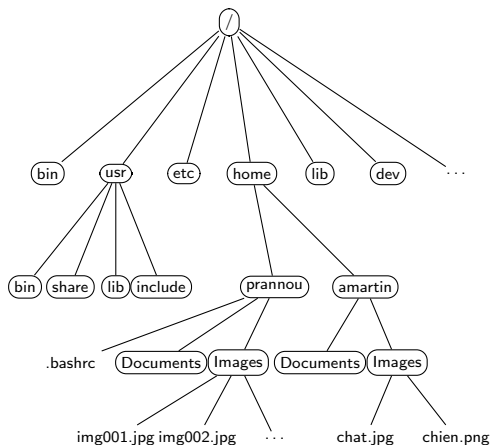
## 4 Réseaux

# Arborescence des fichiers en Linux





# Dossiers standards



- / : racine
- /bin/ : programmes essentiels (ex : ls)
- /dev/, /sys/ : périphériques, drivers
- /etc/ : fichiers de configuration
- /home/ : répertoires personnels des utilisateurs
- /lib/ : librairies essentielles
- /tmp/ : fichiers temporaires
- /usr/ : progr. et librairies "non-essentiels", doc, données partagées

## Sous Unix/Linux : "Tout est fichier"

- les dossiers/répertoires sont des fichiers particuliers (avec une étiquette 'd')
- les "liens", équivalent des raccourcis sous Windows, également (avec une étiquette 'l')
- Chaque fichier possède :
  - un "inode" (numéro unique représentant le fichier)
  - un nom (chemin d'accès)
  - des propriétés (taille, permissions, date de création, ...)
- Nommage des fichiers :
  - Noms sensibles à la casse (majuscule/minuscule)
  - Éviter d'utiliser des espaces
  - Un fichier commençant par . est "caché"
  - Les extensions de fichier sont seulement indicatives : on peut appeler une image chat.mp3

# Désignation des fichiers

Le terminal s'utilise toujours depuis un dossier (par défaut le home de l'utilisateur courant) : les commandes entrées s'exécutent depuis ce dossier.

## Dossiers particuliers

- `.` : désigne le dossier courant
- `..` : désigne le dossier parent
- `~` : désigne le home de l'utilisateur courant

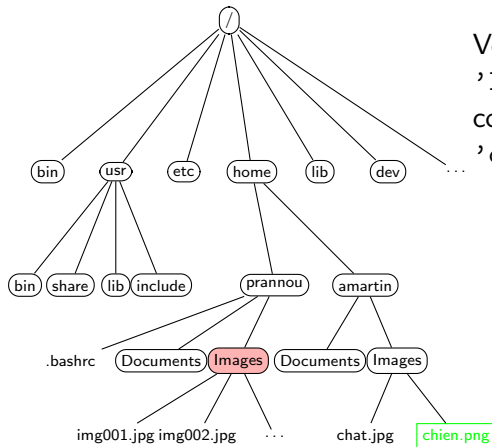
Pour désigner un fichier, on peut utiliser :

- un chemin absolu, depuis la racine, exemple :

`/home/prannou/Images/img001.jpg`

- ou un chemin relatif, par exemple, depuis le dossier `/home/prannou/Documents/` :

`../Images/img001.jpg`



Vous êtes dans le dossier  
'Images' de 'prannou',  
comment désignez-vous le fichier  
'chien.png' ?

# Commandes liées à la navigation

```
$ cd nom_dossier
```

→ change le répertoire d'exécution courant en `nom_dossier`.

```
$ ls
```

→ liste les fichiers et dossiers présents dans le répertoire courant.

```
$ pwd
```

→ Affiche le chemin absolu du répertoire courant.

# Commandes liées à la manipulation de fichiers/dossiers 1

```
$ mkdir nom_dossier
```

→ crée un dossier `nom_dossier` (par défaut dans le répertoire courant)

```
$ touch nom_fichier
```

→ crée un fichier `nom_fichier` (par défaut dans le répertoire courant)

```
$ rmdir nom_dossier
```

→ supprime le répertoire `nom_dossier` (Le répertoire doit être vide)

```
$ rm nom(s)_fichier(s)
```

→ supprime les fichiers `nom(s)_fichier(s)`

## Commandes liées à la manipulation de fichiers/dossiers 2

```
$ cp nom_fichier_source nom_fichier_destination  
$ cp nom(s)_fichier(s)_source nom_dossier_destinat
```

→ crée une copie nom\_fichier\_source en l'appelant  
nom\_fichier\_destination

→ ou crée une copie des fichiers nom(s)\_fichier(s)\_source (avec le même nom) dans le répertoire nom\_dossier\_destination

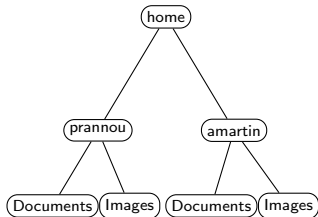
```
$ mv nom_fichier_source nom_fichier_destination  
$ mv nom(s)_fichier(s)_source nom_dossier_destinat
```

→ renomme nom\_fichier\_source en nom\_fichier\_destination

→ ou déplace les fichiers nom(s)\_fichier(s)\_source (avec le même nom) dans le répertoire nom\_dossier\_destination



## Exercice



Vous êtes dans le dossier 'prannou'.

Créez le dossier 'test' dans ce répertoire.

Créez un fichier 'coucou.txt' dans le dossier 'test'.

Copiez le répertoire 'test' dans 'amartin'.

Supprimez le fichier 'coucou.txt' dans le dossier 'test' de 'amartin'.



# Commandes liées à l'affichage des fichiers 1

```
$ cat nom_fichier
```

→ retourne le contenu du fichier `nom_fichier` dans le terminal

```
$ less nom_fichier
```

→ affiche le contenu du fichier `nom_fichier` dans un environnement similaire à `man`

```
$ nano nom_fichier
```

→ ouvre le fichier `nom_fichier` en édition dans le terminal

## Commandes liées à l'affichage des fichiers 2

```
$ head nom_fichier
```

→ affiche le début du fichier `nom_fichier` dans le terminal (par défaut les 10 premières lignes)

```
$ tail nom_fichier
```

→ affiche la fin du fichier `nom_fichier` dans le terminal (par défaut les 10 dernières lignes)

## 1 Systèmes d'exploitation

## 2 Commandes du terminal

- Généralités sur le terminal
- Fichiers et dossiers
- **Utilisateurs**
- Entrées/Sorties et redirections
- Processus
- find et grep

## 3 Scripts

## 4 Réseaux

# Utilisateurs et permissions sous Linux

Sous Linux :

- tous les programmes sont lancés par des utilisateurs
- les utilisateurs ne correspondent pas nécessairement à des humains
- on peut regrouper les utilisateurs pour leur donner globalement des permissions
- tous les fichiers (et dossiers) appartiennent à un utilisateur **et** à un groupe

Les fichiers (et dossiers) ont des droits (permissions) divisés en 3 :

- les droits de l'utilisateur propriétaire
- les droits des membres du groupe propriétaire
- les droits des autres utilisateurs

Chacun de ces droits comporte 3 champs :

- le droit en lecture
- le droit en écriture (modification)
- le droit d'exécution (lancer le fichier comme un programme)

# L'utilisateur root

- appelé "super-utilisateur"
  - peut réaliser toutes les actions sur l'ordinateur (notamment s'attribuer les droits de lecture/écriture/exécution sur tous les fichiers)
  - seul utilisateur pouvant ajouter des utilisateurs
- On peut lancer des commandes "en tant que" root grâce à la commande :

```
$ sudo commande arguments
```

→ lance commande arguments en tant que root

*Remarque : tous les utilisateurs n'ont pas accès à cette commande, seulement ceux appartenant au groupe sudo*

# Commandes liées à la gestion des permissions

```
$ whoami
```

→ affiche l'utilisateur courant

```
$ groups
```

→ affiche les groupes auxquels appartient l'utilisateur courant

```
$ id
```

→ affiche l'utilisateur et ses groupes avec leurs identifiants

```
$ chmod [options] nom(s)_fichier(s)
```

→ change les permissions des fichiers nom(s)\_fichier(s)

## 1 Systèmes d'exploitation

## 2 Commandes du terminal

- Généralités sur le terminal
- Fichiers et dossiers
- Utilisateurs
- Entrées/Sorties et redirections
- Processus
- find et grep

## 3 Scripts

## 4 Réseaux

## Chaque programme possède :

- une entrée standard (par défaut le clavier)
- une sortie standard (par défaut l'écran)
- une sortie d'erreur (par défaut l'écran)

## Que l'on peut rediriger avec le shell :

- entrée standard depuis un fichier : `prog < input-file`
- sortie standard vers un fichier (écrase) : `prog > fichier`
- sortie standard vers un fichier (ajoute à la fin) : `prog >> fichier`
- sortie d'erreur vers un fichier : `prog 2> fichier_err`
- sortie d'un programme vers l'entrée d'un autre : `prog1 arg1 | prog2`



# Commandes élémentaires pour les entrées/sorties

```
$ echo texte
```

→ affiche le texte `texte` sur la sortie standard

```
$ read NOM_VAR
```

→ lit un texte sur l'entrée standard et le stocke dans `NOM_VAR`



## Exercice

On suppose qu'on est dans un dossier contenant un fichier  
'exemple.txt'

Qu'effectue la suite de commandes suivantes :

```
$ echo "10 f l :" > resume.txt  
$ head exemple.txt >> resume.txt
```

## 1 Systèmes d'exploitation

## 2 Commandes du terminal

- Généralités sur le terminal
- Fichiers et dossiers
- Utilisateurs
- Entrées/Sorties et redirections
- **Processus**
- `find` et `grep`

## 3 Scripts

## 4 Réseaux

## Définition 2.1

Un **processus** est une instance d'un programme en cours d'exécution.

Un processus utilise des ressources :

- *code qui s'exécute dans le CPU,*
- *données du processus en mémoire,*
- *autres ressources (port, fichiers ouverts, ...).*

Un processus a des attributs :

- *identifiant, identifiant du processus parent,*
- *propriétaire,*
- *priorité,*
- *commande/programme lancé.*

Remarque :

- Un même programme peut tourner plusieurs fois sous la forme de plusieurs processus.

# Commandes liées aux processus 1

```
$ commande &
```

→ permet de lancer un processus en arrière-plan dans un sous-shell

```
$ ps [option]
```

→ affiche les processus lancés sur l'ordinateur

```
$ pidof commande
```

→ affiche l(es) identifiant(s) du(des) processus lancé(s) pour commande

## Commandes liées aux processus 2

```
$ pstree
```

→ affiche l'arborescence des processus

```
$ kill processus_id
```

→ envoie un signal d'arrêt au processus d'identifiant processus\_id

## 1 Systèmes d'exploitation

## 2 Commandes du terminal

- Généralités sur le terminal
- Fichiers et dossiers
- Utilisateurs
- Entrées/Sorties et redirections
- Processus
- **find** et **grep**

## 3 Scripts

## 4 Réseaux

```
$ find dossier_a_explorer criteres
```

→ cherche tous les dossiers/fichiers dans le dossier dossier\_a\_explorer qui vérifient les critères de recherche criteres

Exemples de critères de recherche :

- par nom : `find . -name "*.txt"` → renvoie tous les fichiers/dossiers terminant par `.txt`
- par type : `find . -type d` → renvoie tous les sous-dossiers de `.`
- par taille, par heure de modification, ...



```
$ grep motif nom(s)_fichier(s)
$ grep motif
```

→ affiche et surligne les lignes des fichiers `nom(s)_fichier(s)` (ou de l'entrée standard) contenant des expressions correspondant au motif `motif`  
Les motifs sont des *expressions régulières*, par exemple :

- `[0-9]+` : motif reconnaissant les nombres entiers
- `ab*a` : motif reconnaissant les mots contenant un `a` puis un certain nombre de `b` et qui terminent par `a` : par exemple `abba`, `abbbbbba`, `aa`, ...

## 1 Systèmes d'exploitation

## 2 Commandes du terminal

## 3 Scripts

- Variables, Entrées/Sorties
- Conditionnelles
- Boucles
- Arithmétique
- Fonctions

## 4 Réseaux

## Motivation

Dans le shell, il arrive qu'on doive répéter plusieurs fois certaines séries d'opérations.

Exemple :

- créer un home avec dossiers (Documents, Images, ...) pour chacun des nouveaux étudiants
- effectuer un traitement (renommage, recadrage, ...) sur une série d'images
- ...

Réécrire à la main toutes les opérations est fastidieux.

## Solution

Il est possible d'automatiser ces tâches, en utilisant des **scripts** !

# Qu'est-ce qu'un script ?

## Définition 3.1

*Un **script** shell est un fichier (au format texte) contenant une ou plusieurs commandes qui seront exécutées de manière séquentielle dans un terminal.*

Caractéristiques :

- l'extension généralement utilisée pour ces fichiers est `.sh`
- la première ligne de ces fichiers devra toujours être (pour des scripts bash) :  
`#!/bin/bash`
- dans une ligne, ce qui suit un `#` est un commentaire

# Exécution d'un script

Il existe plusieurs façons d'exécuter un script.

## ① Méthode 1 :

- rendre le fichier de script exécutable :

```
$ chmod +x nom_fichier_script
```

- lancer le script :

```
$ ./nom_fichier_script
```

## ② Méthode 2 :

- lancer le programme `bash` sur le script :

```
$ bash nom_fichier_script
```

Il est également possible de :

- 1 définir des fonctions dans un fichier :

```
nom_fichier  
  
function nom_fonction1 (){ ... }  
  
function nom_fonction2 (){ ... }
```

- 2 charger ce fichier avec source :

```
$ source nom_fichier
```

- 3 utiliser directement les fonctions du fichier :

```
$ nom_fonction2 arguments
```

- Les instructions sont exécutées les unes après les autres, elles sont séparées :
  - soit par un " ; " :  
`commande1 ; commande2`
  - soit par un retour à la ligne :  
`commande1`  
`commande2`
- `exit n` (où `n` est un nombre) permet d'arrêter un script, par convention `exit 0` signifie que tout s'est bien déroulé dans le script.
- `$(commande)` permet de récupérer (et manipuler) la sortie d'une commande. Exemple :  
`head -n 6 $(tail -n 8 fichier.txt)`

1 Systèmes d'exploitation

2 Commandes du terminal

3 Scripts

- Variables, Entrées/Sorties
- Conditionnelles
- Boucles
- Arithmétique
- Fonctions

4 Réseaux



## Nommage et identification

- assignation : VAR=Valeur (**Pas d'espaces !!!**)
- utilisation : \$VAR
- Identifiants :
  - sans espaces
  - ne commencent pas par un chiffre
  - pas de caractères spéciaux sauf "\_"
  - en général en majuscule.
- Valeurs :
  - toujours de type chaîne de caractères
  - si contiennent des espaces, doivent être contenues dans " "

Qu'affichent les exécutions des scripts suivants :

<pre>V1=coucou echo V1</pre>	
<pre>V1="hello world!" echo \$V1</pre>	
<pre>V1=hello V2=\$V1 world! echo \$V2</pre>	



## Exercices 2

Qu'affichent les exécutions des scripts suivants :

<pre>V1=hello V2 = "\$V1 world!" echo \$V2</pre>	
<pre>V1=coucou V2="\$V1 les gens" echo \$V2</pre>	
<pre># piège V1=hello V2='\$V1 world!' echo \$V2</pre>	

## Définition 3.2

Les **variables d'environnement** sont des variables définies pour le shell en cours d'utilisation et héritées par tous les shells ou processus enfant.

Ce type de variable est utilisé lorsque l'on souhaite en définir pour les commandes/programmes lancés depuis le shell courant.

## Accès et définition

- La commande `env` permet d'afficher toutes les variables d'environnement actuelles.
- La commande `export VAR=Valeur` permet de définir une nouvelle variable d'environnement.

*Remarque : à l'ouverture du shell, des variables d'environnement (comme `PATH`) sont automatiquement chargées, et à la fermeture, elles sont oubliées.*

# Variables spéciales

Dans un script :

- \$0 : nom du script en cours d'exécution
- \$1, \$2, ... : arguments donnés au script
- \$# : nombre d'arguments (le nom du script n'est pas compté)
- \$@ : liste de tous les arguments

## Exercice

Qu'affichent les exécutions des scripts suivants :

<code>script1.sh</code>  <code>echo \$2</code>	<pre>\$ ./script1.sh hi! les gens</pre>	les
<code>script2.sh</code>  <code>echo \$@</code>	<pre>\$ ./script2.sh hi! les gens</pre>	hi! les gens



Qu'affichent les exécutions des scripts suivants :

<code>script3.sh</code> <code>echo "\$#"</code>	<pre>\$ ./script3.sh hi! les gens</pre>	
<code>script4.sh</code> <code>echo \$0</code>	<pre>\$ ./script4.sh hi! les gens</pre>	
<code>script4.sh</code> <code>echo \$0</code>	<pre>\$ bash script4.sh hi!</pre>	

## 1 Systèmes d'exploitation

## 2 Commandes du terminal

## 3 Scripts

- Variables, Entrées/Sorties
- **Conditionnelles**
- Boucles
- Arithmétique
- Fonctions

## 4 Réseaux

- Condition simple :

```
if test
then
    liste de commandes
fi
ou
if test ; then
    liste de commandes
fi
```

- Condition avec alternative :

```
if test ; then
    liste de commandes 1
else
    liste de commandes 2
fi
```

- Condition avec multiples alternatives :

```
if test1 ; then
    liste de commandes 1
elif test2 ; then
    liste de commandes 2
else
    liste de commandes 3
fi
```



## Syntaxe

Un test en bash s'écrit :

```
if [ test ]; then ...
```

**Attention !** aux espaces autour des crochets [ et ].

## Opérateurs

- ET logique : [ test1 ] && [ test2 ]
- OU logique : [ test1 ] || [ test2 ]
- négation : ! [ test ] ou [ ! test ]

## Tests sur les chaînes

- non vide : [ -n chaîne ] ou [ chaîne ]
- égalité : [ chaîne1 = chaîne2 ] (*espaces non nécessaires*)
- différence : [ chaîne1 != chaîne2 ]

## Tests sur les fichiers/dossiers

- existence : [ -e nom\_fichier ]
- est un dossier : [ -d nom\_fichier ]
- est un fichier : [ -f nom\_fichier ]

## Tests sur les nombres

- égalité : [ chaîne1 -eq chaîne2 ]
- inférieur : [ chaîne1 -lt chaîne2 ]
- inférieur ou égal : [ chaîne1 -le chaîne2 ]
- *idem supérieur avec* : -gt -ge



## Exercice

Qu'affichent les exécutions des scripts suivants :

```
V1=3
V2=5
if [ $V1 -lt V2 ]; then
    echo chat
else
    echo chien
fi
```

```
V1=1
V2=2
V3=4
if [ $V2 -gt $V1 ] && [ $V3 -le 4 ]; then
    echo chat
else
    echo chien
fi
```



## Exercice

Écrivez un script qui :

- vérifie qu'un seul argument lui a été donné (sinon affiche une erreur et sort avec la sortie 1)
- vérifie si l'argument est un nom de dossier qui existe :
  - si oui : ne fait rien
  - si non : le crée

## 1 Systèmes d'exploitation

## 2 Commandes du terminal

## 3 Scripts

- Variables, Entrées/Sorties
- Conditionnelles
- **Boucles**
- Arithmétique
- Fonctions

## 4 Réseaux

## Boucle for

```
for VAR in liste_valeur ; do
    liste commande
done
```

## Boucle while

```
while test ; do
    liste commande
done
```

*Remarque : les boucles `while` sont moins utilisées que dans les langages de programmation classiques*



## Exercice

Qu'effectuent les scripts suivants :

```
FILES=$(ls $1)
```

```
for VAR in $FILES; do  
    echo "content : $VAR"  
done
```

```
LINES=$(cat $1)
```

```
for VAR in $LINES; do  
    mkdir $VAR  
done
```

## 1 Systèmes d'exploitation

## 2 Commandes du terminal

## 3 Scripts

- Variables, Entrées/Sorties
- Conditionnelles
- Boucles
- **Arithmétique**
- Fonctions

## 4 Réseaux



Il est possible d'effectuer des calculs sur les entiers en bash, en utilisant la commande :

```
$[calcul]
```

## Exemple

```
$ echo $[1+4*5]  
21
```

Les opérateurs autorisés sont : +, -, \*, /, %



## Exercice

Écrivez un script permettant d'afficher (ligne par ligne) un compte à rebours de 10 à 0.

## 1 Systèmes d'exploitation

## 2 Commandes du terminal

## 3 Scripts

- Variables, Entrées/Sorties
- Conditionnelles
- Boucles
- Arithmétique
- Fonctions

## 4 Réseaux

- Déclaration :

```
function nom_fonction() {  
    liste commandes  
}
```

- Accès aux arguments : \$1, \$2, ...
- Valeur de retour (ou sortie prématurée) avec `return` (convention : 0 si tout se passe bien, autre sinon)
- Utilisation dans le script :

```
nom_fonction argument1 argument2 ...
```

## Particularité des variables

- par défaut, les variables déclarées dans des fonctions sont **globales**, i.e. accessibles en dehors de la fonction.
- déclaration de variable locale :

```
local VAR=Value
```



## Exercice

Écrivez une fonction `affiche_entre` qui :

- vérifie qu'on lui donne en paramètre 3 arguments,
- vérifie que le premier argument est un nom de fichier qui existe,
- affiche le fichier \$1 entre les lignes \$2 et \$3 (incluses),

1 Systèmes d'exploitation

2 Commandes du terminal

3 Scripts

4 Réseaux

- Architecture des réseaux
- Modèles en couches
- Couche Réseau
- Couche Internet
- Couche Transport
- Couche Application
- Programmation réseau

## Définition 4.1

Un **réseau** est un ensemble d'entités (objets, personnes, machines, etc.) interconnectées les unes avec les autres.

Exemples :

- réseau de transport (ferroviaire, routier)
- réseau commercial
- réseau de neurones
- réseau social

## En informatique

Un **réseau informatique** est un ensemble d'équipements (terminaux) reliés entre eux pour échanger des informations.

- communication (personnes, processus, ...)
- partage de ressources (fichiers, applications, matériels, ...)
- ...

## Champs d'études liés aux réseaux

- Architecture des réseaux
- Protocoles de communication
- Algorithmique répartie
- Cybersécurité
- Web
- ...



1 Systèmes d'exploitation

2 Commandes du terminal

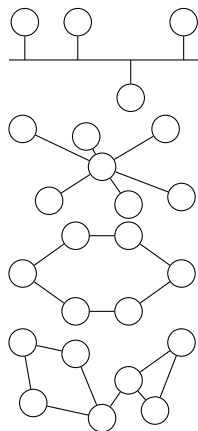
3 Scripts

4 Réseaux

- Architecture des réseaux
  - Modèles en couches
  - Couche Réseau
  - Couche Internet
  - Couche Transport
  - Couche Application
  - Programmation réseau

Un réseau peut être organisé :

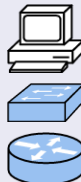
- En bus : tous les terminaux sont reliés à une même ligne de transmission (bus).
- En étoile : tous les terminaux sont reliés à un concentrateur (hub) (*Exemple : réseau domestique autour d'une box*).
- En anneau : les terminaux sont situés sur une boucle.
- En mailles : les terminaux sont reliés en pair à pair sans hiérarchie centrale.



## Terminaux

Parmi les terminaux d'un réseau, on peut distinguer notamment :

- les ordinateurs
- les commutateurs (*switchs*)
- les routeurs



## Liaisons physiques

Parmi les liaisons possibles entre les terminaux, on peut notamment citer :

- les câbles ethernet
- la fibre optique
- le Wifi
- la 4G

On distingue 3 types de réseaux :

- LAN (Local Area Network - Réseau local). Exemple : réseau domestique, d'une petite entreprise.
- MAN (Metropolitan Area Network - Réseau métropolitain). Exemple : réseau d'université, d'établissement scolaire.
- WAN (Wide Area Network - Réseau étendu). Exemple : internet.

Dans un réseau, les données transitent par *paquets*.

Les paquets sont généralement composés de deux parties :

- une **en-tête** (*header*), contenant des informations sur :
  - le protocole utilisé pour la communication,
  - le destinataire du paquet
  - l'émetteur du paquet (selon le protocole)
  - ...
- des données (parfois fractionnées en plusieurs paquets).

Les terminaux du réseau utilisent les informations des en-têtes pour acheminer les paquets vers les bons destinataires.

1 Systèmes d'exploitation

2 Commandes du terminal

3 Scripts

4 Réseaux

- Architecture des réseaux
- **Modèles en couches**
- Couche Réseau
- Couche Internet
- Couche Transport
- Couche Application
- Programmation réseau

## Problématique

Comment faire pour acheminer des données d'un ordinateur à un autre dans un réseau ?

## Définition 4.2 (Protocole)

*Un **protocole** est une spécification de plusieurs règles pour un échange de données entre deux entités.*

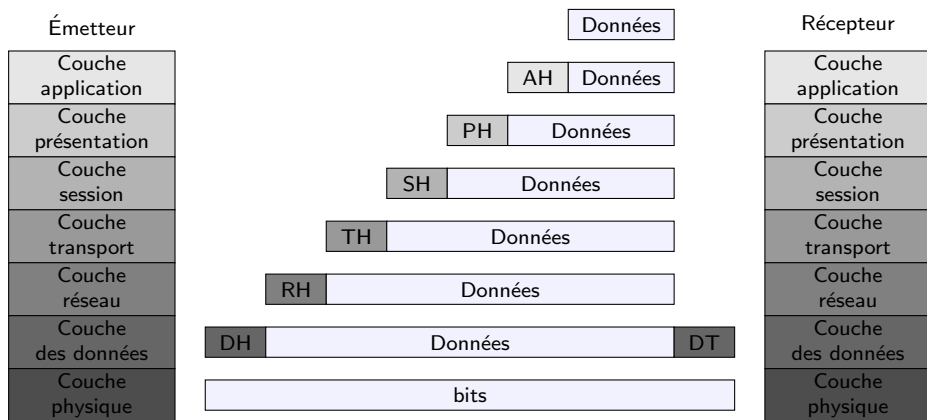
Un protocole définit notamment :

- le format des données (Ex : 1 octet pour le type de protocole, 3 octets pour l'adresse du destinataire, ...),
- les règles d'échange (Ex : L'émetteur envoie un premier message d'authentification, le receveur renvoie un message pour accuser réception, ...)



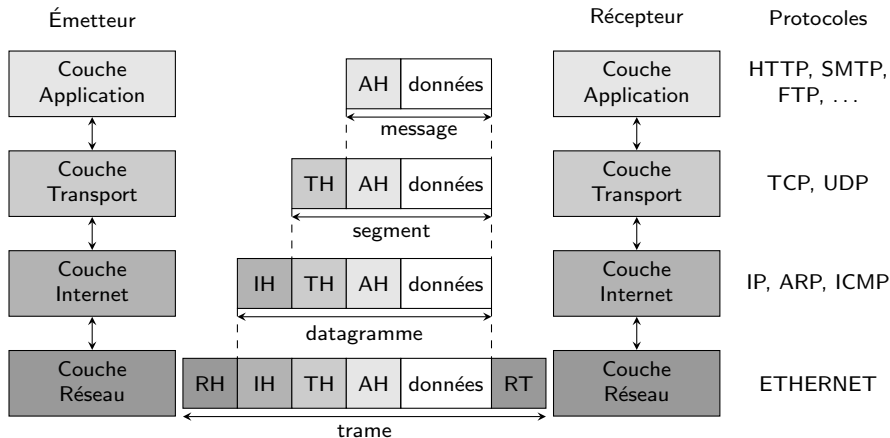
# Modèle OSI

Le modèle OSI (*Open Systems Interconnection*) est un modèle **théorique** qui propose un découpage en 7 couches du réseau. Chaque couche est destinée à accomplir un ensemble de tâches spécifiques.



# Modèle TCP/IP

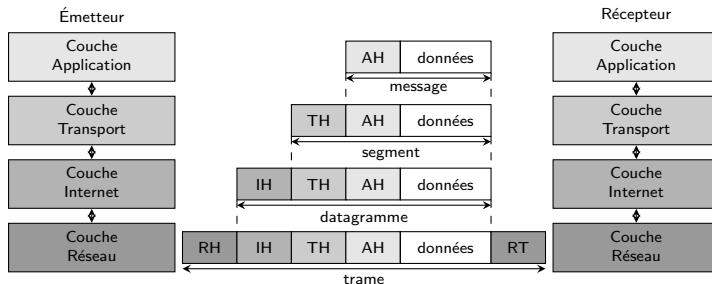
Le modèle TCP/IP est le modèle **utilisé effectivement** pour internet. Il peut (presque) se voir comme une spécialisation du modèle OSI :



# Modèle TCP/IP : rôles des différentes couches

- Couche réseau :
  - Acheminement des données sur la liaison
  - Coordination de la transmission de données (synchronisation)
- Couche internet :
  - Adressage IP
  - Acheminement de datagrammes
  - Gestion de la fragmentation et assemblage
- Couche transport :
  - Transport fiable de segments (en mode connecté)
  - Gestion des erreurs, séquençement, etc.
- Couche Application :
  - Protocole de haut niveau (mail, transfert de fichier, etc.)

# L'encapsulation

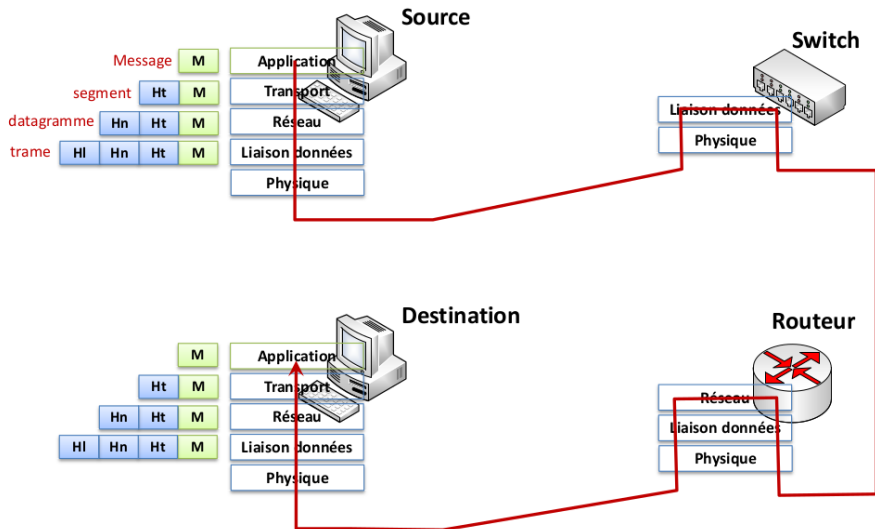


## Définition 4.3

**L'encapsulation** est le procédé consistant à inclure les données d'un protocole dans un autre protocole.

Lors des passages par les différents nœuds du réseau, les paquets ne sont pas toujours entièrement décapsulés.

# L'encapsulation



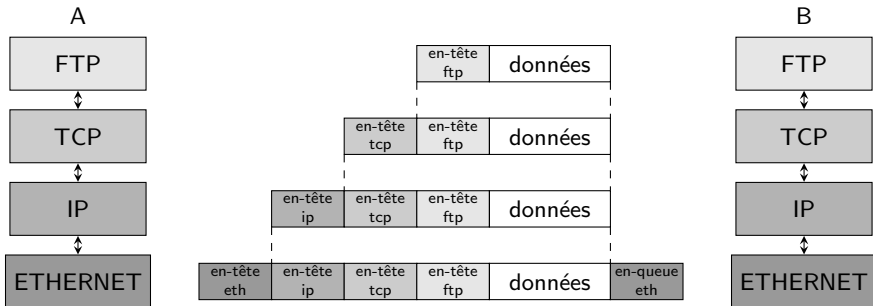
## Définition 4.4

*L'**efficacité du transfert** est le ratio entre la quantité de données utiles et la quantité de données totale envoyée dans le réseau.*

$$\text{Efficacité du transfert} = \frac{\text{données utiles}}{\text{données totales}}$$



# Exemple



A veut envoyer 1024 octets de données à B en utilisant le protocole FTP. On suppose que :

- l'entête FTP a une taille de 70 octet
- l'entête TCP a une taille fixe de 20 octets
- l'entête IP a une taille fixe de 20 octets
- l'entête plus l'en-queue des trames ETHERNET est de 18 octets
- Taille max d'une trame Ethernet est de 1518 octets

Quelle est l'efficacité du transfert ?

1 Systèmes d'exploitation

2 Commandes du terminal

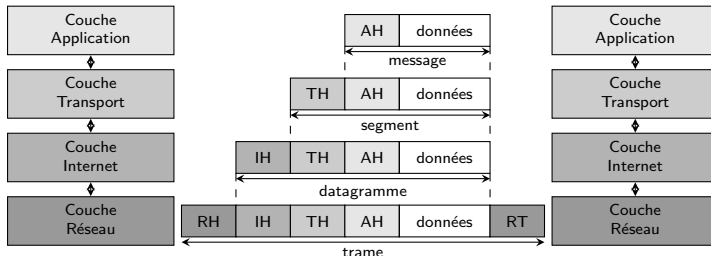
3 Scripts

4 Réseaux

- Architecture des réseaux
- Modèles en couches
- **Couche Réseau**
- Couche Internet
- Couche Transport
- Couche Application
- Programmation réseau



# Couche Réseau



## Fonctionnalité : communication entre machines reliées

- Adressage MAC
- Acheminement de *trames*
- pas de garanties sur la bonne livraison des données.

Protocole de la couche : Ethernet

## Définition 4.5

Une **adresse MAC** (ou **adresse physique** est une suite de 48 bits (6 octets), identifiant de manière unique une carte ou une interface réseau. Elle est généralement représentée en hexadécimal, (Exemple `5E:FF:56:A2:AF:15`)

Remarque :

- L'adresse MAC d'une carte ou interface réseau est fixée par le constructeur mais peut être changée par l'utilisateur.

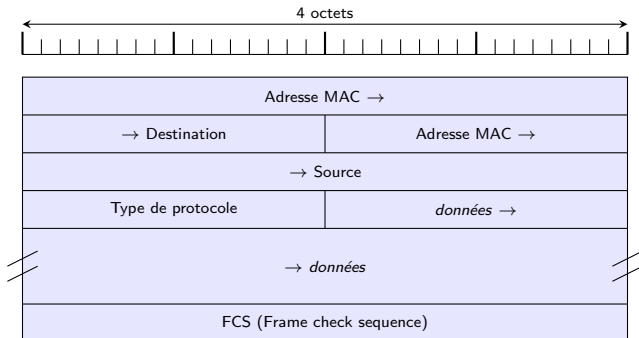


## Exercice

Combien y a-t-il d'adresses MAC possibles ?

$2^{48} \approx 280\,000$  milliards d'adresses possibles

# Format d'une trame Ethernet



- Adresse MAC Destination : adresse de la machine réceptrice (6 octets)
- Adresse MAC Source : adresse de la machine émettrice (6 octets)
- Type de protocole : code selon le protocole encapsulé
  - 0x0800 : IPv4, 0x86DD : IPv6, ...
- FCS : somme de contrôle pour vérification de la bonne transmission

## ip

ip est une commande du terminal permettant d'afficher (et de modifier) les adresses MAC et IP des cartes réseaux d'une machine.

Utilisation :

```
$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> ...
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    ...
2: enp3s0: <NO-CARRIER,BROADCAST,MULTICAST,UP> ...
3: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> ...
    link/ether 00:1b:b1:28:cc:41 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.16/24 brd 10.0.0.255 ...
```

- link/ether est suivi de l'adresse MAC
- inet est suivi de l'adresse IP

1 Systèmes d'exploitation

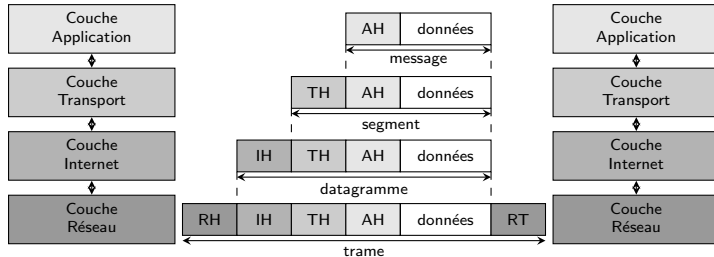
2 Commandes du terminal

3 Scripts

4 Réseaux

- Architecture des réseaux
- Modèles en couches
- Couche Réseau
- **Couche Internet**
- Couche Transport
- Couche Application
- Programmation réseau

# Couche Internet



## Fonctionnalité : communication entre machines

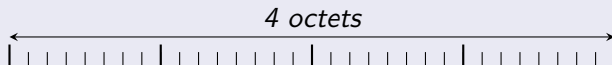
- Adressage IP
- Gestion de la fragmentation et assemblage
- Acheminement de *datagrammes*

## Protocole de la couche

- IP - *Internet Protocol*
- ICMP - *Internet Control and error Message Protocol*
- ARP - *Address Resolution Protocol*
- ...

## Définition 4.6

Une **adresse IP** (v4) est une suite de 32 bits (4 octets) permettant d'identifier une machine sur un réseau.



Elle est généralement représentée en décimal :  $X.Y.Z.T$ , où  $X$ ,  $Y$ ,  $Z$  et  $T$  sont des entiers entre 0 et 255.

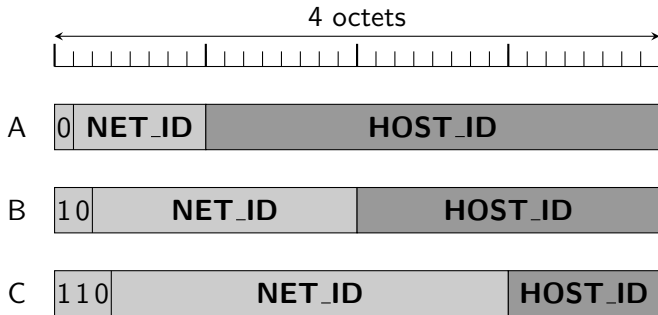
Remarques :

- Les adresses IP sont distribuées par ICANN (*Internet Corporation for Assigned Names and Numbers*), société à but non lucratif.
- Une adresse IP peut être divisée en deux champs :
  - l'identifiant du réseau (**NET\_ID**)
  - l'identifiant de la machine sur le réseau (**HOST\_ID**).



Combien y a-t-il d'adresse IP possibles ?

Les réseaux sont divisés en classe, avec le début des **NET\_ID** fixés :

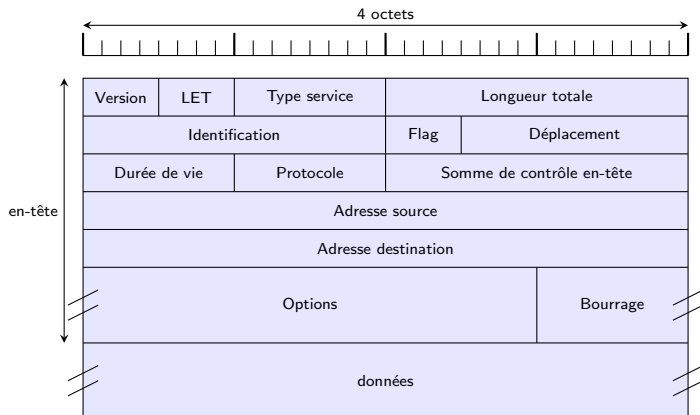


Combien d'adresses possibles dans chacun de ces réseaux ?



- Dans le réseau **NET\_ID**, l'adresse **NET\_ID1...1** est réservée pour le *broadcast* (diffusion à tous les ordinateurs du réseau). Exemple : dans le réseau 192.168, l'adresse de *broadcast* est 192.168.255.255.
- L'adresse 127.0.0.1 (*localhost*) permet un rebouclage local sur la machine.
- L'adresse 0.0.0.0 (utilisée par certains protocoles, route par défaut dans les routeurs)

# Format d'un datagramme IP



- Version : IPv4 ou IPv6
- LET : Longueur En-Tête (en mots de 32 bits)
- Longueur totale : taille totale en-tête + données (en octets)
- Protocole : type du protocole de niveau supérieur

- Adresse source : IP de la machine source
- Adresse destination : IP de la machine destination

Quelle est la taille maximale d'un datagramme IP ?

## Problématique

Les protocoles dans lesquels s'encapsule IP (Ex : Ethernet) ne permettent pas d'envoyer des trames de 60000 octets. (Ex : Trame maximale de 1500 octets pour Ethernet)

Solution : la fragmentation !

- Gérée au niveau des routeurs
- Utilise les champs "Déplacement", "Frag" et "Identification" pour pouvoir reconstruire les paquets.

## Définition 4.7

Le **routage** est le mécanisme par lequel les données d'un équipement émetteur sont acheminées jusqu'à leur destinataire.

Le routage est une tâche dédiée aux **routeurs**.

## Routeur

- Dispositif permettant de choisir le chemin que les datagrammes vont emprunter.
- Possède plusieurs cartes réseau dont chacune est reliée à un réseau différent.
- Utilise la **table de routage** qui définit le chemin à emprunter pour une adresse donnée.

## Rôle

Faire la correspondance entre les adresses IP et MAC d'un même sous-réseau.

Il fait le lien entre la couche Internet et la couche Réseau du modèle TCP/IP.

Principe :

- Chaque ordinateur garde une table de correspondance entre adresse IP et MAC.
- Lorsqu'il manque une correspondance, l'émetteur envoie une requête ARP en *broadcast* contenant l'IP de la cible, la cible répond et l'émetteur met à jour sa table.

## ping

ping est une commande du terminal permettant de tester l'accessibilité d'une machine distante (avec le protocole ICMP).

Elle permet également de mesurer la performance de la connexion (mesure du temps d'aller-retour).

Utilisation :

```
ping -c n ADRESSE
```

- Par défaut la commande ping envoie des paquets jusqu'à ce qu'on l'arrête avec Ctrl+C.
- L'option `-c n` permet de spécifier le nombre de paquets à envoyer avant l'arrêt de la commande.

## ping

traceroute est une commande du terminal permettant d'observer les sauts entre routeurs pour parvenir à la destination.

Utilisation :

```
traceroute ADRESSE
```

- Par défaut la commande traceroute utilise un protocole particulier qui n'est pas toujours autorisé.
- L'option -I permet de spécifier d'utiliser le protocole ICMP (utilisé par ping) pour effectuer son calcul.



1 Systèmes d'exploitation

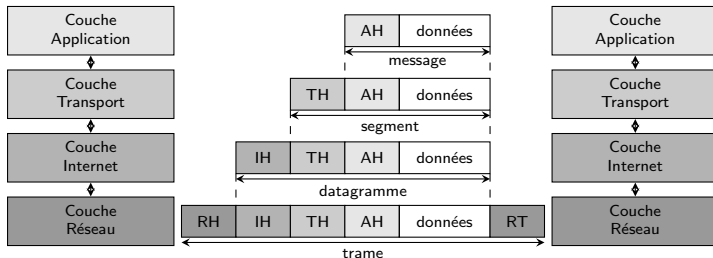
2 Commandes du terminal

3 Scripts

4 **Réseaux**

- Architecture des réseaux
- Modèles en couches
- Couche Réseau
- Couche Internet
- **Couche Transport**
- Couche Application
- Programmation réseau

# Couche Transport



## Fonctionnalités

- Crée un circuit de communication logique entre les **applications** s'exécutant sur des hôtes distants.

## Couche transport vs couche réseau

- Couche Transport : Transfert de données entre **applications**
- Couche Réseau : Transfert de données entre **machines**

# Deux protocoles principaux

## TCP

- réception des segments dans l'ordre
- contrôle de congestion
- contrôle de flot
- mise en place de connexion

## UDP

- sans garantie d'ordre
- sans connexion
- permet le multicast (envoi simultané vers plusieurs récepteurs)

## Définition 4.8 (port logiciel)

*En informatique, un **port** (logiciel) est un numéro, codé sur 16 bits, qui permet de distinguer différents 'interlocuteurs'.*

*Ces 'interlocuteurs' sont des programmes informatiques qui écoutent ou émettent des informations sur ces ports.*



## Exercice

Combien y a-t-il de ports sur un ordinateur ?  $2^{16} = 65\,536$  ports possibles.

Les ports sont classés en 3 catégories :

- du 0 au 1023 : ports utilisés pour les services réseaux les plus courants (Ex : 80 port HTTP)
- du 1024 au 49 151 : ports enregistrés, assignés par l'IANA (ICANN)
- à partir du 49 152 : ports dynamiques : utilisables pour envoyer des requêtes TCP ou UDP.

# Protocole UDP (*User Datagram Protocol*)

## Principe du service

- Pas de connexion entre l'émetteur et le récepteur.
- Chaque segment est traité indépendamment.
- Service "au mieux", les segments UDP peuvent être :
  - perdus,
  - délivrés dans le désordre,
  - pas de contrôle de flux ni d'erreur.

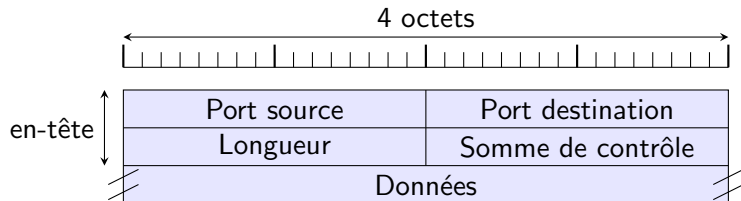
## Avantages

- Pas de délais de connexion
- Transmission rapide

## Utilisation

- DNS (obtention de l'adresse IP à partir d'un nom)
- Multimédia

# Format d'un segment UDP



- Port source : numéro de port de l'application émettrice du segment.
- Port destination : numéro de port de l'application destinataire.
- Longueur : longueur de l'en-tête + données (longueur maximale des données :  $2^{16} - 1 = 65\ 535$  octets).
- Somme de contrôle : code de contrôle d'erreurs

# Protocole TCP

## Principe du service

- Initialisation et fin d'une communication
- Communication en mode connecté
- Récupération des erreurs par réémission
- Re-assemblage des données dans le bon ordre
- Vérification du flot de données

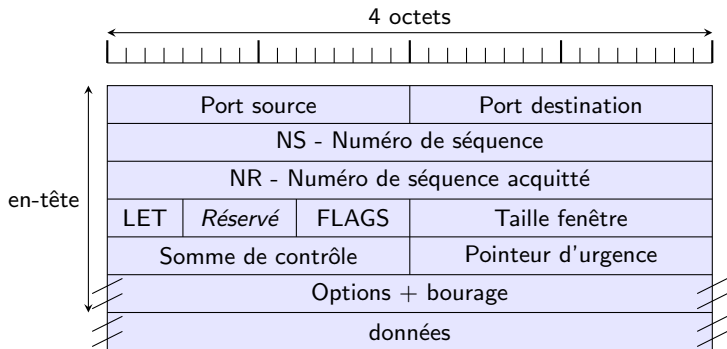
## Avantages

- Arrivée garantie des données
- Pas de saturation du réseau

## Utilisations

- 90% d'internet
- HTTP/HTTPS, FTP, SMTP (mails), ...

# Format d'un segment TCP



- Port source / Port destination
- LET : Longueur de l'En-Tête (en mots de 32 bits).



1 Systèmes d'exploitation

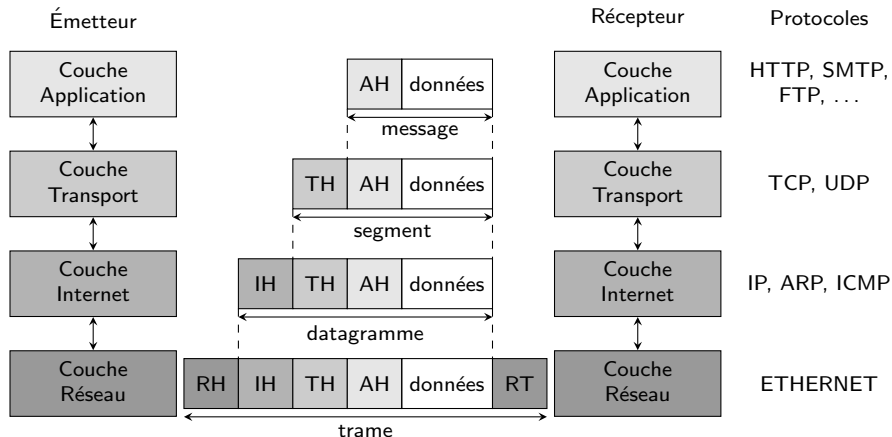
2 Commandes du terminal

3 Scripts

4 Réseaux

- Architecture des réseaux
- Modèles en couches
- Couche Réseau
- Couche Internet
- Couche Transport
- **Couche Application**
- Programmation réseau

# Rappel modèle TCP/IP



# Protocole DNS

## Problématique

Au niveau de la couche Internet, les terminaux sont identifiés par leur adresse IP.

→ pas pratique : les utilisateurs préfèrent des noms symboliques que des nombres (URL).

## Protocole DNS

Le protocole DNS (*Domain Name System*) permet de déterminer des correspondances noms/adresses (et d'autres choses).

## Utilitaire système

La commande `host` permet d'effectuer des requêtes DNS directement depuis le terminal :

```
$ host www.google.com
www.google.com has address 172.217.18.196
www.google.com has IPv6 address 2a00:1450:4007:817
```

## Protocole ssh

ssh (*Secure SHell*) définit à la fois le protocole et le logiciel permettant de se connecter à un terminal d'une machine distante sur laquelle on possède un compte utilisateur.

## Caractéristiques

- Utilise le port 22
- Connexion TCP
- Utilise un chiffrement asymétrique (RSA)

## Commande

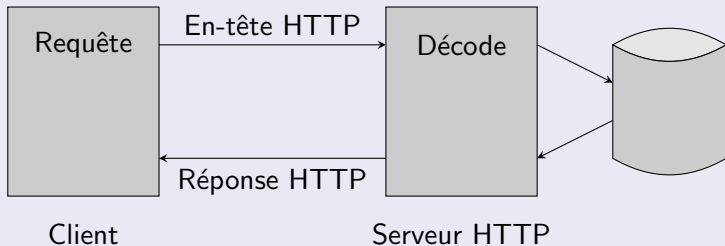
```
$ ssh -l <login> <adresse serveur>
```

# Protocole HTTP

## Caractéristiques

- HTTP : *HyperText Transfer Protocol*
- Utilisé pour le transfert de fichiers (essentiellement au format HTML – *HyperText Markup Language*)
- Utilise le port 80
- Encapsulé dans TCP

## Principe de fonctionnement



## Structure

- 1 Ligne de requête : méthode, URL et version d'HTTP

Exemple : GET /index.html HTTP/1.1

Méthode classiques :

- GET (obtenir une ressource)
- POST (envoi de données)
- DELETE (demande de suppression)

- 2 Lignes d'en-tête : informations supplémentaires sur la requête et/ou le client.

Host: <url>

User-Agent: *infos sur le navigateur*

Accept-Language: *Langage attendu par le navigateur ...*

- 3 Lignes de Corps (*si méthode envoyant des données*)

Exemple:

```
GET /index.html HTTP/1.1
```

```
Host: www.perdu.com
```

```
Accept-Language: FR-fr
```

## Structure

- 1 Ligne de statut : version, code de statut, signification du code  
Exemples : HTTP/1.1 200 OK ou HTTP/1.1 404 Not Found  
Code de statut : valeur numérique (xxx) décrivant le statut de la réponse.
  - 1xx : non utilisé
  - 2xx : Succès
  - 3xx : Redirection
  - 4xx : Erreur côté client
  - 5xx : Erreur côté serveur
- 2 En-tête : informations supplémentaires sur la réponse et/ou le serveur
- 3 Corps : Le document demandé

Exemple :

```
HTTP/1.1 200 OK
```

```
Date: Mon, 28 Mar 2022 20:06:54 GMT
```

```
...
```

```
<html><head><title>Vous Etes Perdu ?</title></head><body>...  
</body></html>
```

1 Systèmes d'exploitation

2 Commandes du terminal

3 Scripts

4 Réseaux

- Architecture des réseaux
- Modèles en couches
- Couche Réseau
- Couche Internet
- Couche Transport
- Couche Application
- Programmation réseau



## Définition 4.9 (Application client/serveur)

Une **application client/serveur** est une application faisant appel à des services distants à travers un échange de messages (requêtes) plutôt que par un partage de données (mémoire ou fichiers).

## Définition 4.10 (Serveur)

Un **serveur** est un programme (et par extension une machine) offrant un service réseau.

## Définition 4.11 (Client)

Un **Client** est un programme qui émet des requêtes (ou demande des services).

Il est toujours l'initiateur du dialogue.

# Exemple d'application client/serveur : serveur de fichier

- Les clients demandent des transferts de fichiers (HTML par exemple)
- Comment gérer des demandes simultanées ?
  - → un processus par client connecté !

## Définition 4.12

*Un **socket** est une interface d'accès au réseau. Il permet d'utiliser un **port logiciel** qui lui a été alloué.*

## Caractéristiques

Un socket se caractérise par :

- un numéro de port,
- une adresse IP,
- un protocole.

# Utilisation des sockets en python

## module

L'utilisation des sockets en python s'effectue avec le module socket :

```
import socket
```

Les sockets sont des *objets* python, on peut leur appliquer des *méthodes*, qui sont des fonctions s'appliquant directement sur les objets concernés.

## Déclaration

Socket TCP/IPv4 :

- `m_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)`

Socket UDP/IPv4 :

- `m_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)`

## Ouverture du dialogue

- client : `m_socket.connect( (IP_serveur, Port_serveur) )` (*en mode connecté*)
- serveur :
  - `m_socket.bind( (IP_serveur, Port_serveur) )`
  - `m_socket.listen()` : *écoute du port logiciel*
  - `socket_client, (IP_client, Port_client) = m_socket.accept()` : *acceptation d'une connexion (en mode connecté)*.

## Transfert de données

- en mode connecté :
  - `m_socket.send(données)` : *envoi de données*
  - `data = m_socket.recv(taille_tampon)` : *réception de données*
- en mode non-connecté :
  - `m_socket.sendto( données, (IP_serveur, Port_serveur) )` : *envoi de données*
  - `data = m_socket.recvfrom(taille_tampon)` : *réception de données*

## Clôture du dialogue

- `m_socket.close()`



## Exemple d'utilisation de socket sous python

```
import socket

IP = '129.20.40.4'
PORT = 5005
BUFFER_SIZE = 1024

s = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
s.bind((IP,PORT))

data, addr = s.recvfrom(BUFFER_SIZE)
print("received data:",data)
```

Qu'effectue le script ci-dessus ?

Ce script se situe-t-il du côté client ou serveur ?



## Exemple d'utilisation de socket sous python

```
import socket

IP = '129.20.40.4'
PORT = 5005
MESSAGE = "Hello, World !".encode('utf-8')

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.sendto(MESSAGE, (UDP_IP, UDP_PORT))
```

Qu'effectue le script ci-dessus ?

Ce script se situe-t-il du côté client ou serveur ?



## Exemple d'utilisation de socket sous python

Écrivez un script python simulant un serveur HTTP sur une machine dont l'IP est 133.45.78.9, recevant une requête et affichant cette requête.



## Exemple d'utilisation de socket sous python

Écrivez un script python simulant un client HTTP effectuant la requête de la page `/index.html` (en HTTP 1.0) sur une machine distante dont l'IP est 133.45.78.9 et affichant la réponse.