



ESIR

CUPGE 1

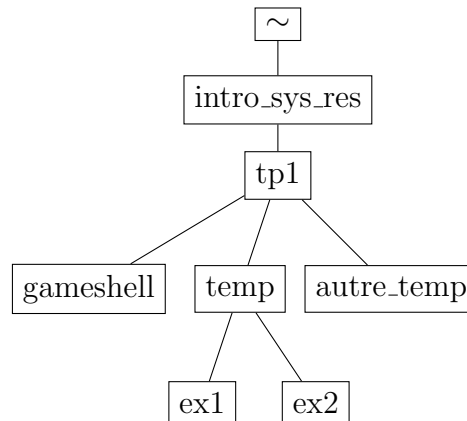
Introduction
aux Systèmes d'exploitation
et Réseaux

Travaux Pratiques : Système

1 Découverte du shell

Exercice 1

1.1. Dans votre **home** (ou si vous utilisez votre machine personnelle : votre dossier personnel), créez (en utilisant uniquement la ligne de commande) l'arborescence de dossiers suivante :



Dans la suite, `~` réfèrera à votre dossier personnel dans lequel vous avez créé cette arborescence.

1.2. Vous vous trouvez dans votre dossier `~`.

- Quelle commande vous permet de vous déplacer dans le dossier `ex2` ? Déplacez-vous y.
- Quelle commande vous permet de vous déplacer dans le dossier `autre_temp` ? Déplacez-vous y.

1.3. Affichez le manuel de la commande `cat`. Que permet de faire l'option `-n` ?

1.4. Entrez la commande :

```
$ cat --help
```

Quelle est la différence avec la commande :

```
$ man cat
```

1.5. Dans le dossier `autre_temp` :

- redirigez la sortie de la commande

```
$ cat --help
```

dans un fichier `cat_help.txt`

- créez un fichier `cd_help.txt`.
- éditez-le avec `nano` et écrivez-y :
Aide pour la commande `cd` :

- d. redirigez la sortie de la commande

```
$ cd --help
```

dans le fichier `cd_help.txt`, à la suite de la ligne que vous venez d'écrire.

1.6. À l'aide des commandes `head` et `tail` :

- affichez les 5 premières lignes du fichier `cd_help.txt`
- affichez les 3 dernières lignes du fichier `cd_help.txt`
- en utilisant une redirection de la sortie de `head` vers l'entrée de `tail`, affichez les lignes 8 à 11 (incluses) de `cd_help.txt`.

On doit obtenir :

La variable `CDPATH` définit le chemin de recherche du répertoire contenant `DIR`. Les noms de répertoires alternatifs dans `CDPATH` sont séparés par un deux-point " : ". Un nom de répertoire vide est identique au répertoire actuel. Si `DIR` commence avec une barre oblique " / ", alors `CDPATH` n'est pas utilisé.

1.7. Renommage et déplacement :

- a. Renommez le fichier `cd_help.txt` en `help_cd.txt`.
- b. Copiez le fichier `cat_help.txt` dans le dossier `ex1`, en lui donnant le nom `help_cat.txt`.
- c. Déplacez tous les fichiers `.txt` du dossier `autre_temp` dans le dossier `ex2`.

Remarque : plutôt que d'écrire le nom de tous les fichiers un à un, on peut les regrouper avec la commande `.txt`.*

1.8. Suppression de fichier et/ou dossier :

- a. Supprimez le dossier `autre_temp`
- b. Supprimer le fichier `help_cd.txt`
- c. en recherchant dans le manuel de `rm`, supprimez en une seule commande le dossier `temp` et tous les fichiers et sous-dossiers qu'il contient.

Exercice 2 Gameshell

Gameshell¹ est un petit jeu de rôle, développé par Pierre Hyvernats² de l'université de Savoie, destiné à faire apprendre au joueur à utiliser le terminal.

2.1. Récupérez le fichier `gameshell.sh` sur moodle, et placez-le dans votre répertoire `gameshell` créé à l'exercice 1.

2.2. Installez les librairies nécessaires au fonctionnement du jeu en tapant la commande :

```
$ sudo apt install gettext man-db procps psmisc nano tree
```

2.3. Placez-vous dans le dossier `gameshell` et lancez le jeu en tapant la commande :

```
$ bash gameshell.sh
```

1. voir <https://github.com/phyver/GameShell>

2. <https://www.lama.univ-savoie.fr/pagesmembres/hyvernats/>

Il y a 42 missions (peut-être davantage depuis la dernière mise à jour) qui vous apprendront à utiliser les différentes commandes vues en cours et d'autres.

Remarques :

- lorsque vous quittez le jeu (en tapant `exit` ou `[Ctrl]+d`), le jeu est sauvegardé et vous pouvez repartir de là où vous étiez en lançant

```
$ bash gameshell-save.sh
```

- **Attention !** si vous relancez le jeu avec la commande de départ, vous perdrez **définitivement** votre progression.

2 Conception de scripts

Exercice 3 TODO Liste

On souhaite implémenter une TODO liste, i.e. une liste de tâches à effectuer, ou d'informations à se rappeler.

Pour ce faire, on utilisera un fichier texte, dans lequel les tâches et informations seront stockées et on les manipulera à l'aide d'un script `bash` pour ajouter, supprimer ou lister l'ensemble des tâches.

3.1. Créez un script contenant une unique fonction `todo` qui offre 3 fonctionnalités :

- lister les tâches en attente
- supprimer une tâche
- ajouter une tâche

Exercice noté à rendre sur moodle avant le **11 février 2024** à 23h59.

- nommez votre fichier `todo.sh`
- commentez bien votre code
- indiquez éventuellement dans les commentaires les parties ne fonctionnant pas
- **Testez votre script !!!** Une plateforme de test est mise en place à l'adresse : tp-tester.esir.univ-rennes1.fr

Exemple d'utilisation

```
$ source todo.sh
$ todo list
1 - finir TP1
2 - téléphoner à tata
3 - inviter Edith à manger
4 - passer la serpillère
$ todo done 1
La tâche 1 (finir TP1) est faite !
$ todo list
1 - téléphoner à tata
2 - inviter Edith à manger
3 - passer la serpillère
$ todo add 2 réviser la chimie
La tâche "réviser la chimie" a été ajoutée en position 2.
$ todo list
1 - téléphoner à tata
2 - réviser la chimie
3 - inviter Edith à manger
4 - passer la serpillère
```

Précisions

- Votre fonction devra analyser son premier argument afin de décider quelle opération effectuer. Il faudra donc se reporter à la section arguments individuels d'une fonction et la description des conditionnelles.
- La liste des tâches doit être sauvegardée dans un fichier (caché) `.todo_list` qui sera stocké dans le dossier personnel de l'utilisateur. Ce fichier contiendra une ligne par tâche, sans numéro.
- Pour simplifier la gestion de ce fichier, il est conseillé de le définir dans une variable au début de votre script :

`TACHES=$HOME/.todo_list`

Remarque : la variable d'environnement `HOME` contient le chemin absolu vers votre dossier personnel.

Votre script devra également gérer les erreurs suivantes :

```
$ todo
La commande 'todo' s'exécute avec 'todo (add | done | list) (arguments)'
```

```
$ todo cuicui
La commande 'todo' s'exécute avec 'todo (add | done | list) (arguments)'
```

```
$ todo add
La commande 'todo add' s'exécute avec 'todo add numero_tache action_a_ajouter'
```

```
$ todo done
La commande 'todo done' s'exécute avec 'todo done numero_tache'
```

Commandes utiles Pour manipuler les tâches, vous pourrez utiliser les choses suivantes :

- La commande `n1` permet d'afficher les lignes d'un fichier, en ajoutant un numéro de ligne. Il est possible d'ajouter un séparateur entre le numéro et la ligne si vous le souhaitez (voir `man n1` pour les détails).
- L'expression `${@:N}` permet de sélectionner tous les arguments à partir du numéro `N`.
- pour ajouter une tâche en position `n`, vous pourrez :
 - ◊ rediriger les `n-1` premières tâches dans un fichier temporaire avec la commande `head` et une redirection `>`,
 - ◊ ajouter la tâche dans le fichier temporaire avec une commande `echo` et une redirection `>>`,
 - ◊ les tâches à partir du numéro `n` dans le fichier temporaire avec la commande `tail` et une redirection `>>`,
 - ◊ remplacer le fichier des tâches par le fichier temporaire.
- pour supprimer la tâche en position `n`, vous pourrez :
 - ◊ rediriger les `n-1` premières tâches dans un fichier temporaire avec la commande `head` et une redirection `>`,
 - ◊ les tâches à partir du numéro `n+1` dans le fichier temporaire avec la commande `tail` et une redirection `>>`,
 - ◊ remplacer le fichier des tâches par le fichier temporaire.

Pour parfaire votre commande, vous pouvez gérer :

- la gestion des erreurs (oubli du numéro de tâche, etc.)
- la gestion de plusieurs fichiers pour des listes différentes.
- un filtre sur les tâches en cours (avec `grep`) pour limiter l'affichage.

Exercice 4 Génération de comptes étudiants

On souhaite pouvoir créer des espaces de travail pour des étudiants à partir d'un fichier texte du type :

```
martin;jeremy;13/04/2001;coucou95@hotmail.com
le reste;nadia;18/01/2001;n_l_r23500@gmail.com
prod'homme;mathis;23/11/2002;iamthebest86@gmail.com
```

4.1. Créez un script permettant, pour chaque étudiant, de :

- Générer un dossier dont le nom correspond au pseudonyme de l'étudiant.
- À l'intérieur de ce dossier, créer un dossier "Documents" et un dossier "Images".
- Générer un fichier texte contenant une seule ligne :

mot de passe :

avec un mot de passe généré aléatoirement.

Exercice noté à rendre sur moodle avant le **18 février 2024** à 23h59.

- nommez votre fichier `admin_etudiants.sh`
- commentez bien votre code
- indiquez éventuellement dans les commentaires les parties ne fonctionnant pas
- **Testez votre script !!!** Une plateforme de test est mise en place à l'adresse : tp-tester.esir.univ-rennes1.fr

Exemple d'utilisation Avec le fichier `liste_etudiants.csv` précédent :

Les passages de ligne ne doivent pas être générés par votre script, ils sont présents uniquement pour la lisibilité.

```
$ ls
admin_etudiants.sh
liste_etudiant.csv
test_home

$ bash admin_etudiants.sh
admin_etudiants.sh s'utilise avec "admin_etudiants.sh liste_nom_etudiants dossier_pa

$ bash admin_etudiants.sh faux_fichier faux_dossier
faux_fichier n'est pas un nom de fichier existant

$ bash admin_etudiants.sh liste_etudiant.csv faux_dossier
faux_dossier n'est pas un nom de dossier existant

$ bash admin_etudiants.sh liste_etudiant.csv test_home/

$ tree test_home/
```



```

test_home/
├── jemartin
│   ├── Documents
│   ├── Images
│   └── mot_de_passe.txt
├── maprodhom
│   ├── Documents
│   ├── Images
│   └── mot_de_passe.txt
└── nalereste
    ├── Documents
    ├── Images
    └── mot_de_passe.txt

9 directories, 3 files

```

Précisions

- Le script prendra 2 paramètres :
 - ◊ le nom du fichier contenant la liste des étudiants,
 - ◊ le nom du dossier dans lequel créer les espaces de travail.
- Le script vérifiera l'existence (et le type fichier/dossier) des arguments.
- Le pseudonyme des étudiants correspond aux deux premières lettres du prénom et aux 7 premières lettres du nom (espaces et ponctuation exclues), on considérera qu'il sera unique au vu de la liste des étudiants.
- Les mots de passe générés devront être de la forme "baki4282", c'est à dire :
consonne voyelle consonne voyelle chiffre chiffre chiffre chiffre

Commandes utiles

- Commande `tr`
- Commande `cut`
- Pour lire une à une les lignes d'un fichier, on peut utiliser la suite de commandes :

```

cat nom_fichier | while read NOM_VARIABLE ; do
    ...
done

```

- Le fichier `/dev/urandom` est un fichier généré pseudo aléatoirement (il change à chaque consultation et fait 32Mo). On peut l'utiliser comme entrée de la commande `tr`, en la combinant avec la commande `head` pour générer des caractères pseudo-aléatoirement.

Exercice 5 Jeu du pendu

Le pendu est un jeu dans lequel un joueur fait deviner des mots à un autre.

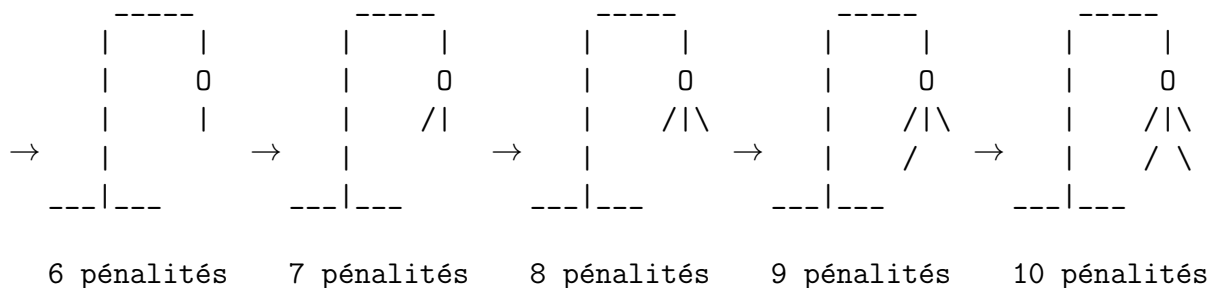
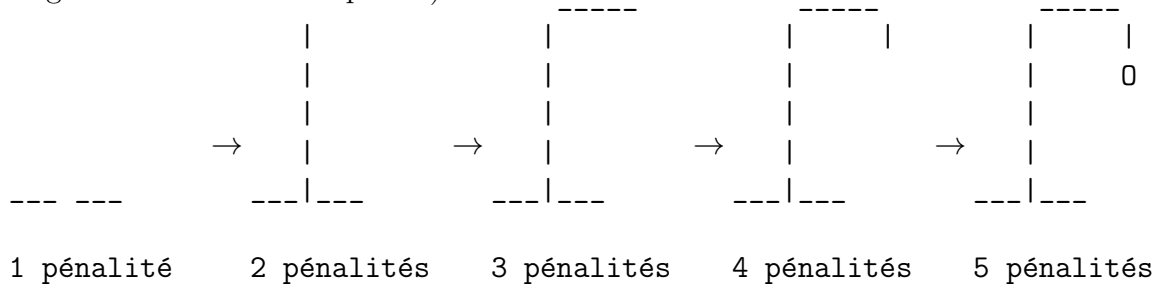
Au départ, le premier joueur indique seulement le nombre de lettres du mot à deviner :

L'autre joueur propose ensuite des lettres (une par une) et :

- si la lettre appartient effectivement au mot, le premier joueur place toutes les occurrences de cette lettre dans le mot :

_E___E

- sinon, marque une pénalité (en général, cette pénalité est représentée par un fragment de dessin d'un pendu) :



5.1. L'objectif de cet exercice est de créer un script qui remplira le rôle du premier joueur.

Votre script devra :

- vérifier qu'un seul argument lui est fourni,
- vérifier que cet argument est bien un nom de fichier,
- choisir un mot au hasard dans ce fichier (on considérera que le fichier contient un mot par ligne, toujours écrit en majuscule),
- le faire deviner.

Exercice noté à rendre sur moodle avant le **25 février 2024** à 23h59.

- nommez votre fichier `pendu.sh`
- commentez bien votre code
- indiquez éventuellement dans les commentaires les parties ne fonctionnant pas
- **Testez votre script !!!** Une plateforme de test est mise en place à l'adresse : tp-tester.esir.univ-rennes1.fr

Exemple d'utilisation

```
$ bash pendu.sh
pendu.sh s'exécute avec un argument :
    pendu.sh nom_fichier
où "nom_fichier" est le dictionnaire pour le choix des mots

$ bash pendu.sh coucou.txt
```

```
"coucou.txt" n'est pas un nom de fichier existant
```

```
$ bash pendu.sh ~/dict.txt
*****

Lettres testées :

-----

Choisissez une lettre :
A
*****

Lettres testées :  A

_A-----

Choisissez une lettre :
B
*****

--- ---

Lettres testées :  A B

_A-----

Choisissez une lettre :
```

...

```
Choisissez une lettre :  
Z  
*****  
  
      _-_-_  
    |         |  
    |         O  
    |         |  
    |         |  
_--|_--
```

Lettres testées :

A B H E O N R S K Z

_A__E_O_SSE

Choisissez une lettre :

R

Lettre choisie incorrecte (o)

_A__E_O_SSE

Choisissez une lettre :

• • •

```

Lettres testées :
A B H E O N R S K Z D Q X

_A__E_O_SSE

Choisissez une lettre :
W

      -----
     |         |
     |         |   O
     |         | / \
     |         | /  \
    ---|-----
```

PERDU :(

Proposition de résolution Votre script pourra être conçu de la manière suivante :

- Définir 6 variables :
 - ◊ `DICT` : représentant le nom du fichier contenant les noms possibles à faire deviner.
 - ◊ `MOT_A_TROUVE` : contenant le mot à trouver,
 - ◊ `MOT_DEVINE` : contenant le mot en cours de recherche (les lettres manquantes étant remplacées par des "_").
 - ◊ `LETTRES_RESTANTES` : initialisée à l'alphabet complet (en majuscule), représentant les lettres qui n'ont pas encore été testées.
 - ◊ `LETTRES_TESTEES` : représentant les lettres déjà testées .
 - ◊ `NB_TENTATIVES` : initialisée à 10, représentant le nombre de tentatives (erreurs) du joueur.
- Définir 4 fonctions :
 - ◊ `choisi_mot`, initialisant la variable `MOT_A_TROUVE` avec un mot choisi aléatoirement dans le dictionnaire, et `MOT_DEVINE`.
 - ◊ `teste_lettre`, teste si la lettre fournie en paramètre appartient au mot, et met à jour `NB_TENTATIVES`, `MOT_DEVINE`, `LETTRES_RESTANTES` et `LETTRES_TESTEES`.
 - ◊ `affiche_pendu`, qui affiche un dessin du pendu en fonction de `NB_TENTATIVES`.
 - ◊ `jeu_pendu`, qui gère la boucle de jeu, dont :
 - l'affichage à l'écran (demande de lettre, mot à deviner, gain/perde, ...)
 - la récupération de la lettre entrée par l'utilisateur (avec vérification qu'elle est correcte)
 - la vérification du nombre de tentatives.

Commandes utiles

- `$RANDOM` : génère un nombre entier aléatoire entre 0 et 32767
- `wc` : permet de compter (nombre de mots, lignes, caractères, ...) des éléments d'une chaîne ou d'un fichier.
- Commande `tr`
- `[[CHAINE1 =~ CHAINE2]]` : teste si `CHAINE1` contient `CHAINE2`

Exercice 6 Gestion des dossiers favoris

Lorsque l'on travaille dans un terminal, certains répertoires reviennent très souvent. C'est souvent le cas du dossier personnel. Le shell offre un raccourci pour se déplacer dans le dossier personnel : la commande `cd` sans argument.

Il n'est par contre pas possible de sauvegarder d'autre répertoires "favoris" pour pouvoir s'y déplacer facilement.

6.1. L'objectif de cet exercice est de créer un petit système de "favoris" pour des répertoires importants.

Votre script devra fournir 4 fonctions :

- **S** (Save) pour sauvegarder un nouveau favori. Cette fonction prend un argument (une chaîne sans espace) et ajoute le répertoire courant dans votre liste de favoris.
- **C** (Change) pour se déplacer dans un répertoire favori. Cette fonction prend un argument (un chaîne sans espace) et cherche dans vos favoris. Si le favori existe, la fonction change le répertoire de travail, sinon, elle ne fait rien.

- R (Remove) pour supprimer un favori de la liste.
- L (List) pour afficher la liste de tous les favoris.

Exercice noté à rendre sur moodle avant le **10 mars 2024** à 23h59.

- nommez votre fichier **favoris.sh**
- commentez bien votre code
- indiquez éventuellement dans les commentaires les parties ne fonctionnant pas
- **Testez votre script !!!** Une plateforme de test est mise en place à l'adresse : tp-tester.esir.univ-rennes1.fr

Exemple d'utilisation

```
$ source favoris.sh
$ pwd
/home/prannou/info202/TP/sujets/TP5
$ S info202-5
Le répertoire /home/prannou/info202/TP/sujets/TP5 est
sauvegardé dans vos favoris.
  -> raccourci : info202-5
$ L
info202-5 -> /home/prannou/info202/TP/sujets/TP5
$ cd
$ pwd
/home/prannou/
$ C info202-5
$ pwd
/home/prannou/info202/TP/sujets/TP5
$ R info202-5
Le favori "info202-5" a été supprimé de votre liste.
$ cd
$ C info202-5
Le favori "info202-5" n'existe pas.
```

Précisions

- Votre script utilisera un fichier (caché) **.favoris_bash** qui sera créé dans votre dossier personnel. Ce fichier contiendra une ligne par favori. Chaque ligne contiendra
 - ◇ le nom du favori (une chaîne sans espace),
 - ◇ la chaîne de caractères '`_->_`',
 - ◇ le chemin absolu du répertoire correspondant.
- Pour simplifier la gestion de ce fichier, il est conseillé de le définir dans une variable au début de votre script :
FAV=\$HOME/.favoris_bash

*Remarque : la variable **HOME** contient le chemin absolu vers votre dossier personnel.*

- Pour manipuler le fichier de favoris, il faudra utiliser les choses suivantes :
 - ◊ La commande **grep** qui permet d'afficher les lignes contenant une chaîne de caractères donnée. Remarque, si on veut imposer que la chaîne cherchée se trouve en début de ligne, on peut la préfixer par le symbole **^** :
 - **grep "abc"** donne toutes les lignes qui contiennent abc,
 - **grep "^abc"** donne toutes les lignes qui commencent par abc.
 - D'autres options de la commande **grep** sont disponibles dans liste des commandes du shell.
 - ◊ les redirections **CMD > FICHIER** et **CMD >> FICHIER** qui permettent de remplacer le contenu de **FICHIER** par le résultat d'une commande, de rajouter le résultat d'une commande à la fin de **FICHIER**.
 Attention : il n'est en général pas possible de rediriger le résultat d'une commande sur un fichier dans le même fichier. Par exemple,
`$ grep "CHAINE" fichier > fichier`
 n'aura pas l'effet attendu car la redirection **> fichier** supprime le contenu avant que la commande **grep "CHAINE" fichier** ne commence.
 Pour faire ceci, il faut donc rediriger la commande dans un autre fichier temporaire, et remplacer ensuite le fichier d'origine par ce fichier temporaire.

Pour parfaire vos commandes, vous pouvez :

- Autoriser les noms de favori incomplets : **C info202** utilisera le favori **info202-5** s'il existe. La commande **C** devra donc :
 - ◊ chercher les favoris "compatibles"
 - ◊ s'il n'y en a qu'un seul, elle l'utilise
 - ◊ s'il y a un favori "exact", elle l'utilise
 - ◊ sinon, elle affiche un message et ne fait rien
- Autoriser une chaîne pour la commande **L** afin de ne lister que les favoris contenant une chaîne de caractères.

Exercice 7 Statistiques sur les fichiers

On souhaite pouvoir afficher des statistiques sur des fichiers :

- taille totale occupée
- taille occupée par les fichiers / répertoires cachés
- nombres de petits (moins de 512kio) et gros fichiers (plus de 15Mio)
- nombre de fichiers et répertoires vides (qui peuvent probablement être supprimés)
- nombres de fichiers Python, ou html, etc.
- ...

7.1. Écrivez un script fournissant une fonction qui analyse le contenu du répertoire courant et de ses sous-répertoires et affiche des statistiques. La quantité de détails dépendra de la valeur de l'argument donné à la fonction :

- 1 (ou pas d'argument) : peu de détails
- 2 : un peu plus de détails
- 3 : beaucoup de détails

Exercice noté à rendre sur moodle avant le **17 mars 2024** à 23h59.

- nommez votre fichier **statistiques.sh**
- commentez bien votre code
- indiquez éventuellement dans les commentaires les parties ne fonctionnant pas
- **Testez votre script !!!** Une plateforme de test est mise en place à l'adresse : tp-tester.esir.univ-rennes1.fr

Exemple d'utilisation

```
$ source statistiques.sh
$ pwd
/home/prannou/02_Sys_Res
$ statistiques
Analyse de /home/prannou/02_Sys_Res :
- 45 répertoire(s)
- 114 fichier(s)
- taille totale : 18M

$ statistiques 2
Analyse de /home/prannou/02_Sys_Res :
- 45 répertoire(s)
  - 1 répertoire(s) caché(s)
  - 0 répertoire(s) vide(s)
- 114 fichier(s) dont
  - 3 fichier(s) caché(s)
  - 1 fichier(s) vide(s)
- taille totale : 18M

$ statistiques 3
Analyse de /home/prannou/02_Sys_Res :
- 45 répertoire(s)
  - 1 répertoire(s) caché(s)
  - 0 répertoire(s) vide(s)
- 114 fichier(s) dont
  - 3 fichier(s) caché(s)
  - 1 fichier(s) vide(s)
  - 87 fichier(s) de moins de 512kio
  - 0 fichier(s) de plus de 15Mio
  - le plus gros fichier est :
    /home/prannou/02_Sys_Res/CM1/Img/RAM_old-plane_
Il y a :
  - 5 fichier(s) Python
  - 17 fichier(s) image
  - 0 fichier(s) vidéo
- taille totale : 18M
```

Précisions Cet exercice est centré sur l'utilisation de `find` (qui permet de calculer la plupart des valeurs affichées).

- les fichiers vidéo à répertorier sont ceux dont l'extension est `.avi`, `.mp4` ou `.mkv`
- les fichiers image à répertorier sont ceux dont l'extension est `.jpg` ou `.png`

Remarque : il n'est pas nécessaire d'écrire des boucles pour cette fonction.

Commandes utiles

- Commande `cut`
- Commande `tr`
- Commande `du`

Commandes utiles

Commande `cut` La commande `cut` permet de récupérer seulement certaines parties d'une chaîne : on précise un délimiteur, les numéros des champs qui nous intéressent. Cette commande agit sur l'entrée standard :

- pour récupérer le deuxième mot (délimiteur : espace)

```
$ echo "Hello World !" | cut -d" " -f2
World
```

- pour récupérer les composantes rouge et bleue d'une couleur RR,GG,BB

```
$ echo "b0,62,bf" | cut -d"," -f1,3
b0,bf
```

Commande `tr` La commande `tr` permet de modifier certains caractères, ou d'en supprimer. Cette commande agit sur l'entrée standard :

- pour changer des caractères on donne simplement deux chaînes

```
$ echo "giraffe" | tr "abcde" "ABCDE"
girAffE
```

Les caractères de la première chaîne sont remplacés par ceux de la deuxième liste. (Il faut que les deux listes aient la même taille.)

- pour changer des caractères en un caractère

```
$ echo "coucou" | tr "co" "h"
hhuhhu
```

- pour supprimer des caractères, on utilise l'option `-d`

```
$ echo "giraffe" | tr -d "abcde"
girff
```


En effet, si on supprime les a, b, c, d et e de la chaîne giraffe, on obtient girff.

- on peut utiliser le complémentaire d'un ensemble (parfois plus simple à écrire), on utilise l'option `-c`

```
$ echo coucou | tr -dc "ou"
ouou
```

Commande du La commande `du` (*disk usage*) prend une liste de fichiers en argument et affiche la taille occupée (en Kio) par chaque fichier. L'option `-c` permet d'afficher également l'espace total utilisé par les fichiers et l'option `-h` permet d'afficher une taille en utilisant des unités "raisonnables" (k, M, G).

```
$ du -ch *.html *.pdf
12K      Fabrication d_un micro-processeur.html
56K      cours.html
2.2M     2I010_2016_support.pdf
12K      survie.pdf
12K      t.pdf
2.3M     total
```

Si on ne lui donne pas d'argument, la commande `du` calcule la taille occupée par tous les fichiers contenu dans le répertoire courant et ses sous-répertoires.

Travaux Pratique : Réseau

3 Découverte des outils réseaux

Exercice 8 `host`

Chaque ordinateur sur Internet doit être identifié par une adresse IP. Par exemple, l'adresse IP (version 4) du serveur qui héberge le site www.google.com est (le 17/03/2022) : 142.250.201.164

Ces adresses sont traduites en adresses IP en utilisant le protocole DNS (Domain Name System). Ceci est fait automatiquement par le navigateur web, mais il est possible de faire une requête DNS directement avec la commande `host`.

8.1. Retrouvez l'adresse IP actuelle de www.google.com et entrez-là dans la barre d'adresse d'un navigateur.

8.2. Quelles sont les adresses IP des sites :

- fr.wikipedia.org
- linuxfr.org

Essayez de vous connecter directement à ces sites avec leur adresses IP.

Remarque : Vous ne pouvez pas accéder à ces sites en utilisant directement leur adresse IP car le serveur utilise le nom de domaine pour décider quelle page afficher. Cela permet d'avoir plusieurs sites web sur une même adresse IP, avec des noms de domaine différents.

Exercice 9 `ping`

Rappel : La commande `ping` permet de mesurer le délai de transmission entre la machine locale et une machine distante. Cette commande s'utilise simplement comme suit :

```
$ ping ADDRESS
```

et envoie des paquets spéciaux vers l'adresse donnée, qui renvoie une réponse. La commande affiche le temps de réponse de chaque paquet, et des statistiques finales (temps de réponse minimum, moyen et maximum).

Remarque : par défaut, la commande continue d'envoyer des paquets jusqu'à ce qu'on la stoppe à la main ("Control-c"). On peut spécifier le nombre de paquets à envoyer avec l'option `-c NBRE_DE_PAQUET_A_ENVOYER`.

9.1. Mesurez le délai de réponse pour atteindre les sites suivants :

- linuxfr.org
- www.univ-rennes1.fr
- www.traveljuneau.com (office de tourisme en Alaska)
- www.monash.edu (université en Australie)

Exercice 10 `traceroute`

Rappel : Les paquets qui voyagent sur Internet sont *routés* : le chemin qu'ils empruntent est décidé dynamiquement par des *routeurs* qui essaient de rapprocher les paquets de leur destination finale.

L'utilitaire `traceroute` permet de visualiser le chemin emprunté par un paquet pour arriver à une destination :

```
$ traceroute ADDRESS
```

*Remarque : par défaut, **traceroute** utilise un protocole qui est parfois bloqué par des pare-feux. Vous pouvez utiliser l'option **-I** pour lui spécifier d'utiliser le protocole ICMP (utilisé par ping).*

10.1. Regardez la route qu'empruntent les paquets pour aller jusqu'aux adresses IP correspondant aux sites suivants :

- linuxfr.org
- www.univ-rennes1.fr
- www.traveljuneau.com
- www.monash.edu

Précisez, le nombre de routeurs empruntés par les paquets.

Exercice 11 Wireshark

Wireshark est un logiciel qui permet de capturer tout ce qui passe par l'interface réseau (carte Ethernet ou carte WiFi) de votre ordinateur et de l'analyser. Cet outil est intéressant pour les administrateurs d'un réseau, mais aussi pour les pirates qui peuvent ainsi écouter le trafic réseau et chercher des informations utiles.

Remarques préliminaires :

- Lorsque vous utilisez Wireshark pour regarder le trafic de votre navigateur, prenez soin de consulter des pages avec le protocole **http** et non **https** (même si la grande majorité des sites internet utilise à présent exclusivement **https** pour des raisons de sécurité).
- Pour éviter que les informations soient "noyées" dans une masse, essayez de n'ouvrir qu'une seule page dans votre navigateur lorsque vous effectuez votre capture.

11.1.

- a. Lancez dans un terminal la commande :

```
sudo wireshark &
```

(le **'&'** permet de garder la main dans le terminal)

- b. Dans *Vue* → *Name Resolution*, cochez toutes les cases (cela rend l'affichage plus intelligible).
- c. Double-cliquez sur la carte réseau que vous souhaitez écouter (Ethernet (**en...**) pour les pc connectés en filaire, Wifi (**wl...** pour les autres) : la capture commence.

Il y a souvent beaucoup de trafic sur votre carte réseau, dont vous n'êtes pas toujours à l'origine (communication avec le réseau local, notamment pour les pc des salles de TP).

11.2. Pour y voir plus clair, Wireshark propose des filtres permettant de n'afficher que les paquets qui nous intéressent.

- a. Dans la barre de filtre, inscrivez **http** puis validez avec 'Entrée'.

- b. Connectez-vous à www.perdu.com dans votre navigateur, et observez les paquets qui ont transité via votre carte réseau.
- c. Double-cliquez sur le paquet contenant la requête `GET` Décrivez l'encapsulation des protocoles de ce paquet.
- d. Double-cliquez sur le paquet contenant la réponse à votre requête. Vérifiez que vous obtenez bien le même contenu qui a été affiché dans votre navigateur.

11.3. Dans la fenêtre principale :

- a. cliquez droit sur l'adresse *www.perdu.com* d'un des paquets `http`. Sélectionnez '*Appliquer comme un Filtre*' → '*Sélectionné*'.
Cela remplace le filtre que vous avez appliqué par :
`ip.src == X.Y.Z.T` ou `ip.dst == X.Y.Z.T`
- b. Remplacez `ip.src` (ou `ip.dst`) par `ip.addr` : cela sélectionnera à la fois les paquets arrivant et sortant vers cette adresse IP.
- c. Observez les 3 paquets `tcp` précédant votre requête `http` : ils représentent la connexion de votre machine au serveur *www.perdu.com* par le protocole `tcp`.
Cet établissement de connexion en 3 temps (`SYN`, `SYN`, `ACK`, `ACK`) s'appelle le *Three-way handshake*.

Exercice 12 netcat

Lorsque vous naviguez sur le web, votre navigateur (qui s'exécute sur votre machine) envoie des *requêtes* à un serveur web.

Pour récupérer des données sur un serveur (par exemple, le contenu d'une page web), il envoie une requête `GET`. Une requête `GET` prend en argument :

- le document demandé
- la version du protocole utilisé
- l'hôte du fichier
- des options (encodage, langue, etc.)

Le serveur web est un processus qui est associé à un port de la machine sur laquelle il s'exécute. Chaque port est associé à un service. Habituellement, les serveurs web utilisent le port 80 (pour HTTP).

La commande `netcat` permet de faire des connections TCP/IP "à la main". Il suffit de spécifier une adresse et un port :

```
$ netcat ADDRESS PORT
```

et :

- les écritures sur l'entrée standard sont automatiquement envoyées au port donné à l'adresse donnée
- les lectures sur la sortie standard sont automatiquement connectées au port donné à l'adresse donnée

Autrement dit, on peut écrire des requêtes sur l'entrée standard, et recevoir les réponses sur la sortie standard.

12.1. Envoi d'une requête "à la main" :

- a. Connectez-vous à l'aide de `netcat` au serveur www.perdu.com sur le port 80 (Utilisez l'option `-C` pour pouvoir écrire une requête ensuite au clavier).
- b. Écrivez la requête suivante :

```
GET /index.html HTTP/1.1
Host: www.perdu.com
```

Attention : terminez votre requête par 2 retours à la ligne.

- `/index.html` est le document que l'on souhaite obtenir,
 - `HTTP/1.1` est la version du protocole HTTP utilisée,
 - `www.perdu.com` est l'hôte contenant le fichier.
- c. connectez-vous via votre navigateur à la page www.perdu.com/index.html et vérifiez que vous obtenez la même page.
 - d. Observez avec Wireshark les requêtes envoyées et reçues lors de cette connexion. Quelles options votre navigateur a-t-il envoyées lors de sa requête ?

Remarque : plutôt que d'écrire 'en direct' la requête, on peut :

- l'écrire dans un fichier qu'on redirige vers l'entrée standard :

```
$ netcat ADDRESS PORT < mon_fichier
```

- ou l'afficher avec la commande `echo` qu'on redirige avec un pipe :

```
$ echo -ne "GET ... \r\n\r\n" | netcat ADDRESS PORT
```

Cela évite notamment de se faire déconnecter du serveur lorsque l'on met trop de temps à écrire la requête à la main.

Exercice 13 mini tchat avec netcat

La commande `netcat` permet de communiquer sur un port, mais également de créer un petit serveur sur un port choisi.

La commande :

```
$ netcat -l -p PORT
```

ouvre le port donné sur la machine locale et écouter ce qui se passe dessus. (Pour des raisons de sécurité, le port doit être supérieur à 1000.)

Pour se connecter à ce serveur, il faut que le client lance :

```
$ netcat ADDRESS PORT
```

où :

- `ADDRESS` est l'adresse IP de la machine sur lequel le serveur est lancé.
- `PORT` est le port choisi lors du lancement du serveur.

13.1. Tchat en local :

- a. Quelle est votre adresse de rebouclage local ?

b. Ouvrez deux terminaux :

- Dans le premier, lancez un serveur avec `netcat`.
- Dans le second, connectez-vous à ce serveur.

c. Vérifiez que lorsque vous écrivez sur l'un des terminaux, le texte s'affiche également dans l'autre.

13.2. Tchat sur le réseau de l'université :

Attention ! : si vous êtes sur votre machine personnelle, connectez-vous en wifi sur le réseau *univ-rennes1*.

a. Déterminez, avec un binôme, qui de vous deux sera le serveur et le client.

b. Déterminez quelle est l'adresse IP du serveur.

c. Choisissez un numéro de port (supérieur à 1000).

d. Lancez `netcat` en mode écoute sur la machine serveur.

e. Lancez `netcat` sur le client en vous connectant au serveur.

f. Vérifiez que lorsque vous écrivez sur l'un des terminaux, le texte s'affiche également dans l'autre.

13.3. Observez avec Wireshark le trafic engendré par `netcat`. Quels sont les protocoles utilisés ?

Exercice 14 `ssh`

Rappel : `ssh` est un logiciel (et un protocole) permettant de se connecter à un terminal d'une machine distante.

14.1. Dans un terminal, connectez-vous en `ssh` à la machine `welcome1.istic.univ-rennes1.fr` avec votre sésame.

Remarque : `welcome1` est une machine du réseau ISTIC/ESIR

14.2. Vérifiez que vous y retrouvez les fichiers et dossiers de votre compte université.

Exercice 15

15.1. À l'aide de la commande `ip addr`, déterminez l'adresse IP de votre machine. Dans quelle plage d'adresse se situent les adresses des machines de votre sous-réseau ?

15.2. Écrivez un script `bash` lançant un `ping` vers les adresses de votre sous-réseau et affichant celles qui sont effectivement attribuées (on pourra se limiter aux 20 premières adresses).

4 Utilisation de python

Exercice 16 Analyse de trames Ethernet

L'objectif de cet exercice est de décoder des trames Ethernet.

Dans un premier temps, vous décoderez des trames qui ont été enregistrées dans des variables (en binaire).

Exercice noté à rendre sur moodle avant le **26 mars 2023** à 23h59.

- rendez un seul fichier nommé `decodeur_trame.py`
- indiquez éventuellement dans les commentaires les parties ne fonctionnant pas
- **Testez votre programme!!!**

Préliminaires

Utilisation de struct Le package `struct` permet de manipuler, et notamment d'appliquer un masque sur un tableau d'octets (*bytes*) pour pouvoir sélectionner les éléments qui nous intéressent.

Exemple : supposons qu'on ait la suite de bits suivante :

```
0000010000000000000000000000000001010000100000110100001101001
```

et que cette suite doive être interprétée de la manière suivante :

```
00000100 000000000000000000000000000001010 00010000 01101000 01101001
-----
entier          entier          entier   char.   char.
```

On peut utiliser le module `struct` pour définir un masque qui permet de séparer la chaîne comme on le souhaite. La commande `unpack` prend en paramètre un *masque* et un tableau d'octet (*bytearray*) et retourne un tuple correspondant à la chaîne "démasquée".

Le masque s'écrit à l'aide d'une chaîne de caractère commençant par `'!'`. Puis on sépare la chaîne en groupe d'octets à l'aide des symboles :

- `B` pour des entiers représentés sur **1** octet
- `H` pour des entiers représentés sur **2** octets
- `L` pour des entiers représentés sur **4** octets
- `ns` pour un tableau de n caractères (sur n octets)

Il existe d'autres symboles pour les masques, voir la doc³ pour une documentation complète.

Dans notre exemple, on souhaite décoder la chaîne en utilisant :

1. un entier sur 1 octet
2. un entier sur 4 octets
3. un entier sur 1 octet
4. 2 caractères

3. Utilisation de struct : <https://docs.python.org/3/library/struct.html>

Le masque que l'on devra appliqué sera donc : `"!BLB2s"`.

Utilisation :

```
import struct

# en python la chaine est représentée en octets, pas en bits
chaine = b'\x04\x00\x00\x00\n\x10hi'

unmask_chaine = struct.unpack("!BLB2s", chaine)
```

À la suite de l'exécution de ce script, `unmask_chaine` contient le tuple : (4, 10, 16, b'hi'). On peut accéder à chacun de ses éléments comme pour un tableau (Ex. `unmask_chaine[0]` ...).

16.1. Le tableau d'octets :

`b'\x00\x01\x00\x02trois\x00\x00\x00\x04\x05\x00\x00\x00\x06'`

contient 6 *champs* :

1. 2 entiers codés sur 2 octets
2. 5 caractères (compte pour 1 champ)
3. 1 entier codé sur 4 octets
4. 1 entier codé sur 1 octet
5. 1 entier codé sur 4 octets

Écrivez un script permettant de retrouver la valeurs des différents champs.

Problème des portions d'octets Le module `struct` ne permet de découper un *bytearray* en portion d'octet (Exemple : 4 bits correspondent à un entier, 4 bits correspondent à un autre). Or dans certains cas, les protocoles spécifient des champs sur moins d'un octet.

Exemple : un octet (*de valeur 106*) contient les bits :

01101010

- Si on souhaite récupérer seulement la valeur des 3 bits de droite (010 = 2), on peut utiliser un modulo :

$$106 \bmod 2^3 = 2$$

- Si on souhaite récupérer seulement la valeur des 5 bits de gauche (01101 = 13), on peut utiliser un *décalage de bits* en supprimant ceux qui ne sont pas utiles (ici 3) :

$$106 \gg 3 = 13$$

16.2. Les 4 premiers bits (à gauche) de l'octet 11101101 (237 en décimal) représente un entier. Utilisez une commande `python` pour récupérer cette valeur.

16.3. Récupérez le fichier `decodeur_trame.py` sur moodle.

Décodage de paquet UDP

16.4. Écrivez une fonction python `decode_UDP` prenant en paramètre un tableau d'octets `data` représentant un datagramme UDP et retournant un couple :

- une chaîne de caractère représentant les informations de l'entête UDP :

```
#####+++_Paquet_UDP_+++
#####Port_source#####:...
#####Port_Destination_:...
#####Longueur_totale_:...
```

- les données contenues dans le segment UDP (après l'en-tête).

Décodage de paquet TCP

16.5. Écrivez une fonction python `decode_TCP` prenant en paramètre un tableau d'octets `data` représentant un segment TCP et retournant un couple :

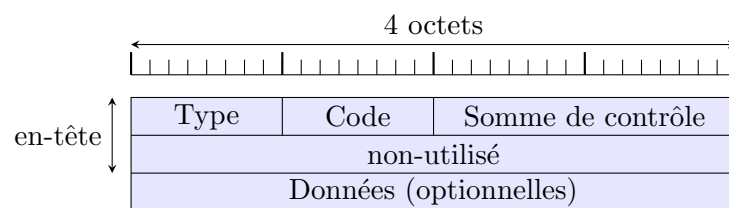
- une chaîne de caractère représentant les informations de l'entête TCP :

```
#####+++_Paquet_TCP_+++
#####Port_source#####:...
#####Port_Destination_:...
#####Longueur_en-tête_:...
```

- les données contenues dans le segment TCP (après l'en-tête).

Décodage de paquet ICMP

*ICMP est le protocole notamment utilisé par la commande **ping**, il est encapsulé dans des datagramme IP.*



Format d'un paquet ICMP

16.6. Écrivez une fonction python `decode_ICMP` prenant en paramètre un tableau d'octets `data` représentant un paquet ICMP et retournant une chaîne de caractère représentant les informations de l'entête ICMP :

```
#####+++_Paquet_ICMP_+++
#####Type_:...
```

Décodage de datagramme IP

16.7. Écrivez une fonction `python decode_adresse_IP` prenant un paramètre un entier codé sur 32 bits représentant une adresse IP et retournant une représentation en chaîne de caractère de cette adresse ($X.Y.Z.T$).

16.8. Écrivez une fonction `python decode_IP` prenant en paramètre un tableau d'octets `data` représentant un datagramme IP et retournant un triplet composé :

- d'une chaîne de caractère représentant les informations de l'entête IP :

```

#####Paquet_IP#####
#####Version#####:...
#####Longueur_en-tête#####:...
#####Protocole#####:...
#####Adresse_source#####:...
#####Adresse_dest.#####:...

```

- du numéro de protocole encapsulé :
 - ◊ 1 pour ICMP
 - ◊ 6 pour TCP
 - ◊ 17 pour UDP
- des données encapsulées.

Décodage de trames Ethernet

16.9. Écrivez une fonction `python decode_adresse_MAC` prenant en paramètre un tableau de 6 octets représentant une adresse MAC et retournant une représentation en chaîne de caractères de cette adresse ($A1:A2:A3:A4:A5:A6$).

Indication : la commande "%02x" % n permet d'obtenir une représentation en hexadécimal de n.

16.10. Écrivez une fonction `python decode_Ethernet` prenant en paramètre un tableau d'octets `data` représentant une trame Ethernet et retournant un triplet composé :

- d'une chaîne de caractère représentant les informations de l'entête Ethernet :

```

>>>Trame_Ethernet<<<
#####Adresse_MAC_Destination#####:...
#####Adresse_MAC_Source#####:...
#####Protocol#####:...

```

- du numéro du protocole encapsulé :
 - ◊ 0x0800 (2048) pour IPv4
 - ◊ 0x86DD (34525) pour IPv6
 - ◊ 0x0806 (2054) pour ARP
 - ◊ ...
- des données encapsulées

Remarques : dans le protocole Ethernet, des données d'en-queue sont rajoutées à la fin de la trame, mais ici on ne les récupérera pas.

Recombinaison

16.11. Écrivez une fonction `decode_trame` prenant en paramètre une trame Ethernet et affichant l'encapsulation des protocoles et les informations (en utilisant les fonctions définies précédemment).

Vous devez être sous linux et disposer des droits administrateurs sur votre machine pour faire la question suivante.

La commande :

```
|| socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(0x0003))
```

permet de définir un socket de niveau 3 écoutant directement sur la carte réseau et récupérant des trames Ethernet.

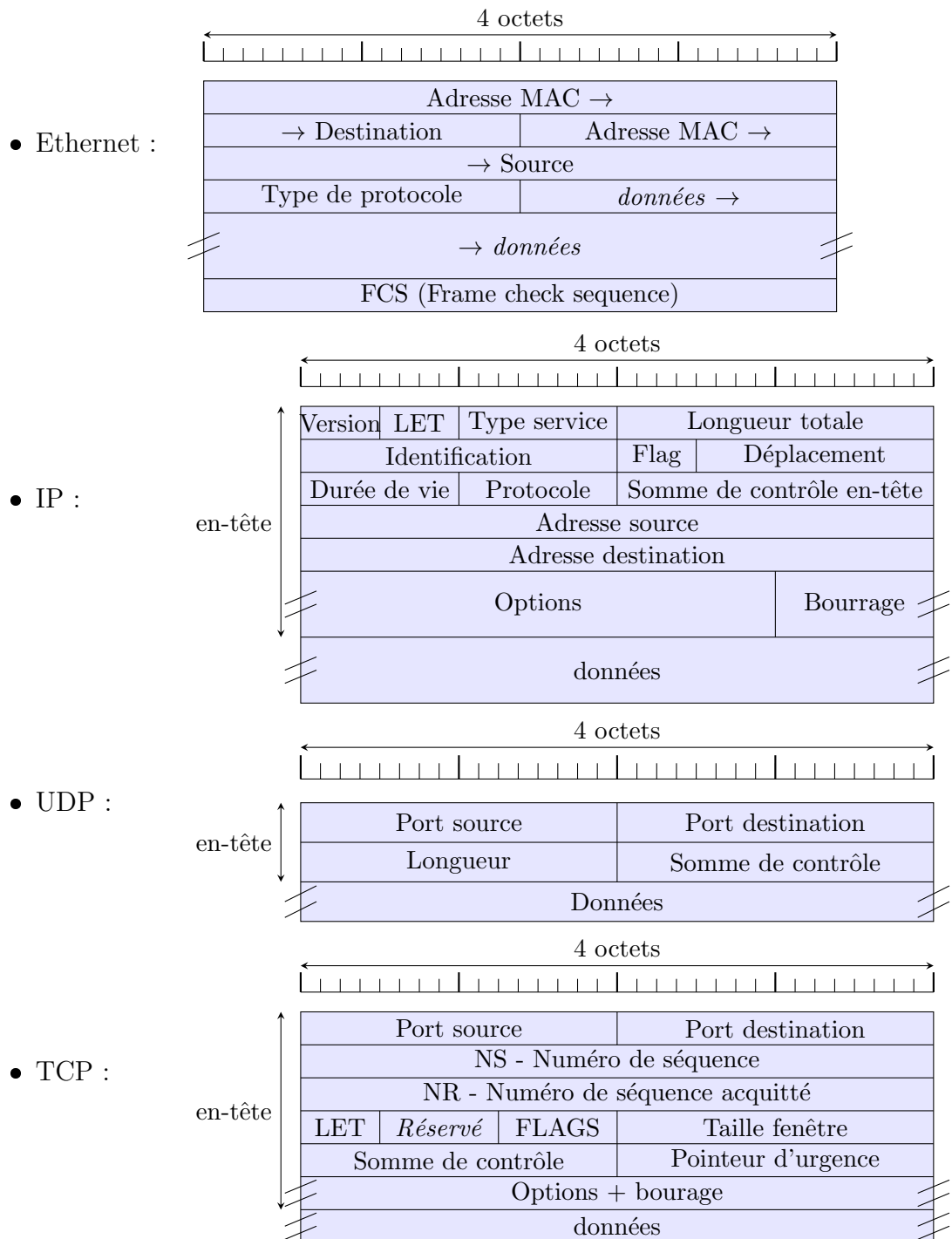
16.12. Définissez un tel socket et faites afficher les informations passant sur votre carte réseau.

Testez votre programme :

- en effectuant un **ping** sur une machine distante (vérifiez les IP)
- en vous connectant à un site internet (en HTTP)

Vérifiez que vous obtenez bien les même informations qu'avec Wireshark.

Rappels sur les format des différents protocoles



Exercice 17 Serveur HTTP

L'objectif de cet exercice est de créer un serveur HTTP qui sera capable de transmettre des données à un navigateur classique (firefox, chrome, ...) le consultant.

Dans un premier temps, les questions seront à faire en local sur votre machine : vous utiliserez 127.0.0.1 comme adresse du serveur.

Exercice noté à rendre sur moodle avant le **02 avril 2023** à 23h59.

- rendez un seul fichier nommé `serveurHTTP.py` (le fichier `clientHTTP.py` n'est pas à rendre),
- indiquez éventuellement dans les commentaires les parties ne fonctionnant pas
- **Testez votre programme!!!**

Préliminaires

17.1. En reprenant les exercices du cours, créez deux fichiers de script python :

- `serveurHTTP.py` qui :
 - ◇ écoute sur le port 8080 ou un autre port s'il est déjà utilisé (*le port 80 est un port réservé : vous ne pouvez pas l'utiliser sans être **root** sur votre machine*),
 - ◇ affiche dans la console les requêtes reçues.
 - ◇ retourne le message de réception "**Bien reçu !**" aux clients qui lui ont envoyé une requête.
(*Votre serveur ne doit pas traiter qu'une seule requête : utilisez une boucle infinie qui écoute en continu le port qui lui est attribué.*)
- `clientHTTP.py` qui :
 - ◇ effectue une requête HTTP sur l'adresse 127.0.0.1 (port 8080) pour la page `/index.html`,
 - ◇ affiche la réponse du serveur à l'écran.
(*Votre client en revanche, n'envoie qu'une seule requête : n'utilisez pas de boucle infinie !*)

Lancez vos scripts (en commençant par le serveur) et vérifiez le bon fonctionnement.

Décodage de la requête HTTP

(Un squelette de code pour `serveurHTTP.py` est disponible sur moodle.)

17.2. Écrivez une fonction python `decode_requete_http` prenant en paramètre une chaîne de caractère représentant une requête HTTP et retournant un couple `page, options` où :

- `page` est la chaîne de caractères représentant la page demandée,
- `options` est un dictionnaire contenant les lignes d'en-tête de la requête.

Par exemple, pour :

```
requete = "GET /page1.html HTTP/1.1\r\nHost: localhost\r\n
Accept-Language: fr-FR,en;q=0.3\r\n
User-Agent: Mozilla/5.0 Firefox/98.0\r\n\r\n"
```

(sur une seule ligne dans votre code)

`decode_requete_http(requete)` doit renvoyer : `"/page1.html"`, dict, avec :

```
dict : {'Host': 'localhost',
       'Accept-Language': 'fr-FR,en;q=0.3',
       'User-Agent': 'Mozilla/5.0 Firefox/98.0'}
```

Commande utiles :

- `chaine.split(c)` permet, à partir d'une chaîne de caractères `chaine`, de récupérer sous forme de tableau tous les champs entre les caractères `c` (les caractères `c` ne sont pas comptabilisés).

Exemples :

```
"Le chat est noir".split(" ") -> ['Le', 'chat', 'est', 'noir']
```

```
"le riz est de retour".split("r") -> ['le ', 'iz est de ', 'etou', '']
```

- une chaîne de caractère est considérée comme un tuple de caractères : on peut utiliser les commandes python manipulant des tuples, notamment :

- ◊ `chaine[3:6]` : permet de récupérer la chaîne composée des caractères 3 (inclus) à 6 (exclus) de `chaine`.

- ◊ `chaine[:8]` : permet de récupérer le début de `chaine` jusqu'au caractère 8 (exclus).

- ◊ `chaine[:-2]` : permet de récupérer `chaine` sans les 2 derniers caractères.

Exemples :

```
"bonjour"[:3] -> 'bon'
```

```
"bonjour"[2:6] -> 'njou'
```

```
"bonjour"[:-1] -> 'bonjou'
```

Construction de la réponse HTTP

Les pages que le serveur peut transmettre sont des fichiers html présents dans des dossiers spécifiques.

17.3. Récupérez le dossier `pages_serveur.zip` et décompressez le dans votre répertoire de travail (dans lequel se trouve `serveurHTTP.py`).

Rappel Une réponse HTTP a le format suivant :

<code>HTTP/1.1 200 OK</code>	-> Ligne de statut (version, code, sign.)
<code>Date: Mon, 04 Apr 2022 09:06:33 GMT</code>	\
<code>Server: Apache</code>	
<code>Upgrade: h2</code>	-> Lignes d'en-tête :
<code>Connection: Upgrade</code>	infos supplémentaires sur la réponse
<code>...</code>	
<code>Content-Type: text/html</code>	/
	-> Ligne vide
<code><html><head><title>Vous Etes Perdu ?...</html></code>	-> Données

Dans cet exercice, on aura deux types de réponse :

- Une réponse "OK" quand la page recherchée existe (et est accessible), dont l'en-tête sera :

```
HTTP/1.0 200 OK
Content-Type:text/html
Content-Length:#à calculer#
```

- et une réponse "404 NotFound" quand la page recherchée n'existe pas, dont l'en-tête sera :

```
HTTP/1.0 404 NotFound
Content-Type:text/html
Content-Length:#à calculer#
```

17.4. Écrivez une fonction `python get_reponse` prenant en paramètre l'url d'une page qu'un client cherche à consulter et retournant :

- si cette page existe, la réponse HTTP contenant le contenu de cette page (en une chaîne de caractères),
- sinon, la réponse d'erreur 404 avec le contenu de la page `pages_serveur/page404.html`.

Précisions :

- N'oubliez pas la ligne vide entre l'en-tête et les données.
- N'oubliez pas d'ajouter un passage à la ligne à la fin des données (et de le prendre en compte dans le calcul de la longueur).

Commandes utiles :

- la commande `fichier = open(nom_fichier,"r")` permet d'ouvrir un fichier existant sur la machine. La commande provoque une erreur si le fichier n'existe pas ou ne peut pas être ouvert.
- la commande `fichier.read()` permet de récupérer le contenu du fichier (comme une chaîne de caractère)
- Il est possible de 'récupérer' des erreurs provoquées par des commandes `python` en utilisant `try : ... except Exception : ... :`

```
try :
    commandes_pouvant_provoquer_une_erreur
except Exception :
    commandes_à_exécuter_en_cas_d'erreur
```

Exemples :

```
◇ a = 8
try :
    a = a/2
    print("pas d'erreur :",a)
except Exception :
    a = a/4
    print("une erreur !!",a)
Provoque l'affichage "pas d'erreur : 4.0".

◇ a = 8
try :
    a = a/0
    print("pas d'erreur :",a)
except Exception :
    a = a/4
    print("une erreur !!",a)
Provoque l'affichage "une erreur !! 2.0".
```


Serveur complet

17.5. Écrivez une fonction python `traite_requete` prenant en paramètre une requête HTTP et :

- qui décode la requête
- si le champ optionnel "Accept-Language" existe et commence par "fr" :
 - ◊ cherche la page recherchée dans le dossier `pages_serveur/fr/`
 - ◊ sinon dans le dossier `pages_serveur/en/`

Exemples :

```
traite_requete("GET /page1.html HTTP/1.1\r\n
Host: localhost\r\n
Accept-Language: fr-FR,en;q=0.3\r\n
User-Agent: Mozilla/5.0 Firefox/98.0\r\n\r\n")
```

(normalement sur une seule ligne)

-> retourne une réponse HTTP avec le contenu de
'pages_serveur/fr/page1.html'

```
traite_requete("GET /pages/index.html HTTP/1.1\r\n
Host: localhost\r\n
Accept-Language: en\r\n\r\n")
```

-> retourne une réponse HTTP avec le contenu de
'pages_serveur/en/pages/index.html'

```
traite_requete("GET /autres_pages/toto.html HTTP/1.1\r\n
Host: localhost\r\n\r\n")
```

-> retourne une réponse HTTP avec le contenu de
'pages_serveur/en/autres_pages/toto.html'

17.6. Intégrez votre fonction `traite_requete` dans le code de votre serveur en le modifiant.

- a. Vérifiez que vous obtenez bien les réponses HTTP attendues avec votre client (page en français, page par défaut en anglais, page erreur 404).
- b. Connectez vous à votre serveur via votre navigateur (entrez dans la barre d'adresse : *adresse_IP :port_utilisé/url_de_la_page*). Vérifiez que vous obtenez les pages en français.
- c. Modifiez les préférences de votre navigateur (dans Firefox : *Paramètres* → *Général* → *Choix de la langue préférée pour l'affichage des pages*), en plaçant 'Anglais [en]' en favori. Vérifiez que les pages envoyées par votre serveur sont en anglais.

17.7. (*non évalué*) En changeant l'adresse ip du serveur par celle de votre machine, vérifier avec d'autres binômes que vous pouvez accéder à des serveurs HTTP distants.

Exercice 18 Mini tchat en UDP

L'objectif de cet exercice est de programmer un mini chat en python qui utilisera le protocole UDP.

Dans un premier temps, les questions seront à faire en local sur votre machine : vous utiliserez 127.0.0.1 comme adresse du serveur.

Exercice noté à rendre sur moodle avant le **09 avril 2023** à 23h59.

- rendez deux fichiers nommés `serveur_chat.py` et `client_chat.py`,
- indiquez éventuellement dans les commentaires les parties ne fonctionnant pas
- **Testez vos programmes!!!**

Exemple d'utilisation

```
$ python3 serveur_chat.py
recv_data : b'__new_name__:pr'
recv_data : b'__new_name__:as'
recv_data : b'coucou'
recv_data : b'hello'
recv_data : b'__new_name__:hs'
recv_data : b'hey'
recv_data : b'je suis l\xc3\xa0 !'
recv_data : b'__quit__'
```

Serveur

```
$ python3 client_chat.py
Enter your name : pr

***pr*** vient de rentrer sur le chat !

***as*** vient de rentrer sur le chat !

[as] : coucou
hello

***hs*** vient de rentrer sur le chat !

[hs] : hey
[hs] : je suis là !
quit
$
```

Client1

```
$ python3 client_chat.py
Enter your name : as

***as*** vient de rentrer sur le chat !

coucou
[pr] : hello

***hs*** vient de rentrer sur le chat !

[hs] : hey
[hs] : je suis là !

***pr*** a quitté le chat.
```

Client2

```
$ python3 client_chat.py
Enter your name : hs

***hs*** vient de rentrer sur le chat !

hey
je suis là !

***pr*** a quitté le chat.
```

Client3

Principe de fonctionnement :

- niveau serveur :
 - ◇ le serveur garde en mémoire (dans un dictionnaire) une correspondance entre (adresse_ip_client, port_udp_client) et "nom_du_client"
 - ◇ le serveur reçoit trois types de message :
 - une première connexion d'un client, dont le message sera de la forme :
`__new_name__:#nom#`
qui entraîne l'envoi à tous les clients connectés (y compris celui-ci) du message :
`***#nom#***_vient_de_rentre_sur_le_chat_!`
 - un message standard de la forme
`__message__:#message_reçu_par_le_serveur#`
qui provoque l'envoi à tous les clients connectés, excepté l'expéditeur, du message :
`[#nom_expéditeur#]_:#message_reçu_par_le_serveur#`
 - un message de départ de la forme :
`__quit__`
qui provoque :
 - l'envoi à tous les clients connectés, excepté l'expéditeur, du message :
`***#nom#***_a_quitté_le_chat.`
 - l'envoi à l'expéditeur du message :
`__quit__`
 - la suppression de l'adresse de l'expéditeur du dictionnaire des adresses.
*Rappel : pour supprimer un élément de clé **cle** d'un dictionnaire **dict**, on utilise :*

```
|| del dict[cle]
```
 - ◇ le serveur affiche dans sa console tous les messages qu'il reçoit sous la forme :
`recv_data_:#message_reçu_non_décodé#`
- niveau client :
 - ◇ au lancement du script, le programme commence par demander un nom à l'utilisateur avec le message :
`Enter_your_name_:`
puis le script envoie un premier message au serveur de la forme :
`__new_name__:#nom#`
avec `#nom#` la chaîne de caractères entrée par l'utilisateur.
 - ◇ le client exécute ensuite deux actions en parallèle (on utilisera deux processus) :
 - un *receiver* qui attend de recevoir des données et les affiche lorsqu'elles arrivent.
 - un *sender* qui attend que l'utilisateur entre une chaîne au clavier avant de l'envoyer au serveur.
 - ◇ lorsque l'utilisateur du client souhaite quitter le chat, il tape `quit` (sans espace avant ou après) dans son terminal, ce qui provoque :
 - l'envoi du message :
`__quit__`
au serveur.

- l'arrêt du *sender* (par la commande **break** dans la boucle infinie).
- ◇ lorsque le client reçoit le message :
`__quit__`
 il arrête le *receiver* (par la commande **break** dans la boucle infinie), et le programme doit terminer.

Préliminaires

18.1. En reprenant les exercices du cours, créez deux fichiers de script python :

- `serveur_chat.py` qui :
 - ◇ écoute sur le port 5005 (pour le protocole UDP) ou un autre port s'il est déjà utilisé,
 - ◇ affiche dans la console les requêtes reçues,
 - ◇ retourne le message de réception "Bien reçu !" aux clients qui lui ont envoyé une requête.
(Votre serveur ne doit pas traiter qu'une seule requête : utilisez une boucle infinie qui écoute en continu le port qui lui est attribué.)
- `client_chat.py` qui :
 - ◇ envoie un paquet UDP (contenant ce que vous voulez) sur l'adresse 127.0.0.1 (port 5005),
 - ◇ affiche la réponse du serveur à l'écran.
(Votre client en revanche, n'envoie qu'une seule requête : n'utilisez pas de boucle infinie !)

Client

18.2. Au début de votre script `client_chat.py`, définissez 3 constantes :

```
BALISE_NEW_NAME = "__new_name__:"
BALISE_MESSAGE = "__message__:"
BALISE_QUIT = "__quit__"
```

18.3. Écrivez une suite de commandes demandant à l'utilisateur d'entrer son nom et envoyant au serveur le message de connexion avec la spécification donnée en introduction.

Comme expliqué en introduction, le script client aura deux tâches à effectuer en parallèle :

- un *receiver* qui attend de recevoir des données et les affiche lorsqu'elles arrivent.
- un *sender* qui attend que l'utilisateur entre une chaîne au clavier avant de l'envoyer au serveur.

18.4. Dans votre script `client_chat.py`, écrivez deux fonctions `send` et `receive` ne prenant pas de paramètre et programmant ces comportements.

Précisions :

- Utilisez des boucles infinies dans les deux fonctions
- Il n'y a aucune mise en forme des messages reçus à faire : elle est faite au niveau du serveur.
- Dans votre fonction `send`, le message envoyé sera soit précédé de `BALISE_MESSAGE` soit de `BALISE_QUIT` (lorsque le message entré est exactement "quit").

Pour lancer les deux processus à partir de votre programme, vous pouvez utiliser le code suivant :

```
def send() :
    ...
def receive() :
    ...

# Création des processus
send_thread = threading.Thread(target=send)
recv_thread = threading.Thread(target=receive)

# Lancement des processus
send_thread.start()
recv_thread.start()
```

Remarque : lorsque vous lancerez vos scripts depuis un terminal, il faudra utiliser Contrôle+C pour arrêter le serveur.

Serveur

Proposition de résolution Pour implémenter votre serveur, vous pouvez :

- a. Définir 3 constantes au début de votre script :

```
BALISE_NEW_NAME = "__new_name__:"
BALISE_MESSAGE = "__message__:"
BALISE_QUIT = "__quit__"
```

- b. Définir (comme variable globale) un dictionnaire **adresses** qui associera des clés (adresse_ip, port) à des valeurs noms (en chaîne de caractère).
- c. Définir une fonction **send_entrance_notification** prenant en paramètre une adresse **addr** (techniquement un couple (adresse_ip, port)) et un nom **name**, et qui :
- vérifie que **addr** n'est pas déjà présente dans **adresses** et ajoute la correspondance **addr -> name**
 - envoie à toutes les adresses de **adresses** le message :
`\n***#nom#*** vient de rentrer sur le chat !\n`
- d. Définir une fonction **send_message** prenant en paramètre l'adresse de l'expéditeur **addr** et le message envoyé **message**, et :
- qui envoie à tous les utilisateurs connectés (dans **adresses**) **sauf** l'expéditeur, le message :
`[#nom_expéditeur#] : message_reçu_par_le_serveur`
- e. Définir une fonction **send_quit_notification** prenant en paramètre l'adresse de l'expéditeur **addr**, et :
- qui envoie à tous les utilisateurs connectés (dans **adresses**) **sauf** l'expéditeur, le message :
`***#nom_expéditeur#*** a quitté le chat.`
 - qui envoie le message `__quit__` à l'expéditeur,
 - qui supprime l'adresse **addr** du dictionnaire.
- f. Définir une fonction **traite_data** prenant en paramètre l'adresse de l'expéditeur **addr** et le message envoyé **data**, et :

- si le message commence par `BALISE_NEW_NAME`, appelle `send_entrance_notification` avec les bons paramètres,
- si le message commence par `BALISE_MESSAGE`, appelle `send_message` avec les bons paramètres,
- si le message **est exactement égal** à `BALISE_QUIT`, appelle `send_quit_notification` avec les bons paramètres.

g. Placer `traite_data` dans votre boucle infinie pour traiter les messages entrant.

18.5. (*non évalué*) En changeant l'adresse ip du serveur par celle de votre machine, vérifier avec d'autres binômes que vous pouvez communiquer avec votre mini-chat sur des machines distantes.

Travaux Pratique : Salle réseau

5 Découverte salle réseau 1

Exercice 19 Utilisation des VM et des switches

Les objectifs de ce TP sont :

- découvrir la manipulation de machines virtuelles Linux,
- découvrir la manipulation des switches,
- vous familiariser avec la procédure d'utilisation de la salle réseau

Pour chacun des TPs en salle machine (sauf celui-ci), vous devrez faire un compte-rendu que vous déposerez au format pdf sur Moodle **au maximum une semaine après la séance**.

Conseil Le temps de TP en salle réseau est limité, il peut donc être utile de ne pas perdre de temps dans la recherche des commandes à effectuer. Aussi je vous suggère de compléter la fiche technique (modèle disponible sur Moodle), voire d'y inscrire d'autres commandes que vous jugerez pertinentes. Cette fiche pourra notamment vous resservir en ESIR1 ou ESIR2 selon la spécialité que vous choisirez.

Description des composants

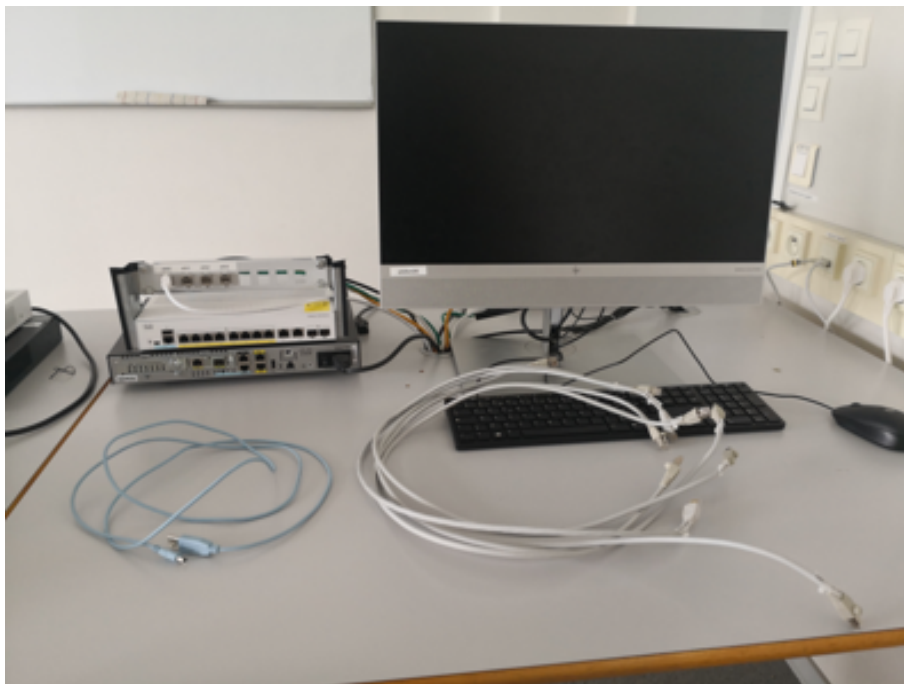







FIGURE 1 – Présentation du poste complet

Les salles de TP réseau contiennent du matériel (cher et fragile) permettant de mettre des machines physiques et/ou virtuelles en réseau. Parmi les différents composants, on distingue notamment :

Poste fixe	
Switch	
Routeur	
cables Ethernet (RJ45)	
cable Intercom	

Poste fixe

Le poste fixe :

- est installé avec le système d'exploitation Debian-Linux (proche de Ubuntu),
- possède 4 interfaces ("prises RJ45")
- possède un compte utilisateur :
 - ◊ login : `usertp`
 - ◊ mot de passe : `!2+en+dur?!`
- possède plusieurs logiciels d'installés :
 - ◊ libreOffice (pour le compte-rendu),
 - ◊ capture d'écran,
 - ◊ wireshark (même si nous l'utiliserons plutôt sur les machines virtuelles)
 - ◊ minicom (pour le dialogue avec les switches et routeurs)
 - ◊ ...

Vous n'avez pas les droits administrateurs sur cette machine !

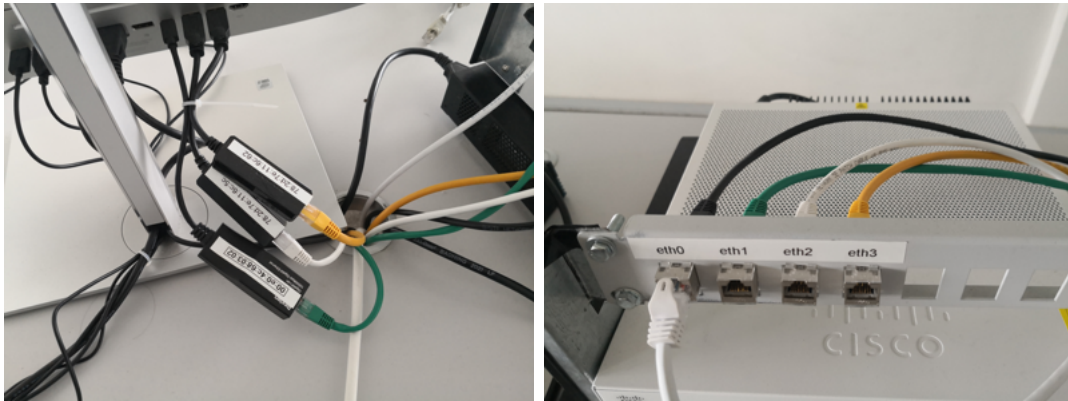
Une grande partie des commandes réseaux nécessitent d'avoir ces droits sur la machine, vous les aurez cependant sur les machines virtuelles (VM).

Faites votre compte-rendu sur le poste fixe, pas sur les VM. La plupart du temps, vos compte-rendus contiendront des captures d'écran des différentes VM, or seul le poste fixe peut 'voir' les VM, les VM ne peuvent se voir entre elles et ne peuvent accéder au poste fixe. Aussi lors de l'enregistrement d'une capture d'écran, il faut qu'elle soit faite à partir du poste fixe.

Rappel adresses IP

- 19.1. De combien d'octets est composée une adresse IP (v4) ? Cela correspond à combien de bits ?
- 19.2. Combien d'adresses IPv4 possibles existe-il ?
- 19.3. Rappelez la commande pour déterminer l'adresse MAC des interfaces présentes sur un PC.
- 19.4. Quels sont les adresses MAC des interfaces du poste fixe ?

Vérification IMPORTANTE ! Normalement, les machines de la salle réseau sont câblées de telle sorte que les interfaces "Eth0", "Eth1", ... correspondent effectivement aux étiquettes placées sous les prises RJ 45



Cependant, il arrive que cet ordre soit perturbé (parce que les utilisateurs précédents ont modifié les branchements, ou à cause d'un bug au lancement du PC).

Une inversion de ces câbles est **difficile et pénible** à détecter après la mise en place. Aussi, lors de chaque TP, il est important de commencer par vérifier cette correspondance.

19.5. Vérifier que les adresses Mac des interfaces "Eth0", ... correspondent bien aux étiquettes des câbles arrivant aux étiquettes "Eth0", ...

Si vous vous apercevez d'un problème, signalez-le à votre encadrant.

Mise en place des machines virtuelles

La place sur les tables étant limitée, il est difficile d'y placer plusieurs écrans. Aussi, nous utiliserons des machines virtuelles (VM) qui permettront de simuler différentes machines physiques.

Cette année, nous n'utiliserons que des machines virtuelles Linux.

Remarque La mise en place des VM n'est pas instantanée (environ 1min30 pour Linux, 3min pour Windows). Aussi, lors des TP, il peut être intéressant de lancer directement l'installation des VM au début, mais cela implique d'avoir lu le sujet avant.

19.6. Suivez la procédure d'installation (normalement présente sur le bureau du poste fixe) pour installer **deux** VM Linux que vous nommerez VM1 et VM2.

19.7. **Important !** Avant de lancer vos VM, assurez-vous d'avoir associé la bonne interface (du poste fixe) à la bonne VM.

19.8. Lancez vos machines, connectez-vous y :

- login : `root`
- mot de passe : `!2+en+dur?!`

19.9. Déterminez les adresses IP de chacune de vos VM ainsi que les adresses MAC de leur interfaces. Sont-elles les mêmes que celles indiquées sur le poste fixe ? La commande `ip addr` permet, en plus d'afficher des informations, de modifier les adresses ip données aux interfaces. Tapez *man ip-address* pour accéder à l'en-semble du manuel.

On notera notamment :

- suppression d'une adresse IP sur une interface :

```
ip addr del ADDRESS/MASK_SIZE dev DEVICE_NAME
```

où :

- ◇ ADDRESS est une adresse IP écrite sous la forme W.X.Y.Z
- ◇ MASK_SIZE est la taille (en nombre de bits) du masque du sous-réseau.
- ◇ DEVICE_NAME est le nom de l'interface (en général `eth0`).
- ajout d'une nouvelle adresse IP sur une interface :

```
ip addr add ADDRESS/MASK_SIZE dev DEVICE_NAME
```

avec les mêmes arguments.

- Par défaut, l'interface n'est pas activée dans la VM, cela est notamment indiqué dans le résultat de la commande `ip addr`, avec la présence du mot clé `DOWN` (pour désactivé) ou `UP` (pour activé).

On active l'interface `DEVICE_NAME` en tapant la commande :

```
ip link set dev DEVICE_NAME up
```

Important !

- Il faut toujours supprimer l'adresse IP existante avant d'en entrer une nouvelle.
- le nom de l'interface est celui présent dans la VM, pas celui du poste fixe.

19.10. En utilisant les commandes précédentes :

- donnez l'adresse `192.168.X.1` à votre première VM (X est votre numéro de poste), avec un masque de 24 bits.
- donnez l'adresse `192.168.X.2` à votre deuxième VM (X est votre numéro de poste), avec un masque de 24 bits.

19.11. Vérifiez que les adresses IP ont bien été modifiées.

Important ! Il est possible que votre configuration manuelle 'saute' (après une mise en veille de la VM, un branchement avec un câble connecté à un réseau, ...), dans ce cas, vous devrez réitérer cette opération.

19.12. Rappelez une commande du terminal permettant de vérifier l'accessibilité d'une machine distante.

19.13. Sans connecter de câble, tester l'accessibilité de la VM1 à partir de la VM2. Les VMs sont-elles connectées ?

Découverte du switch

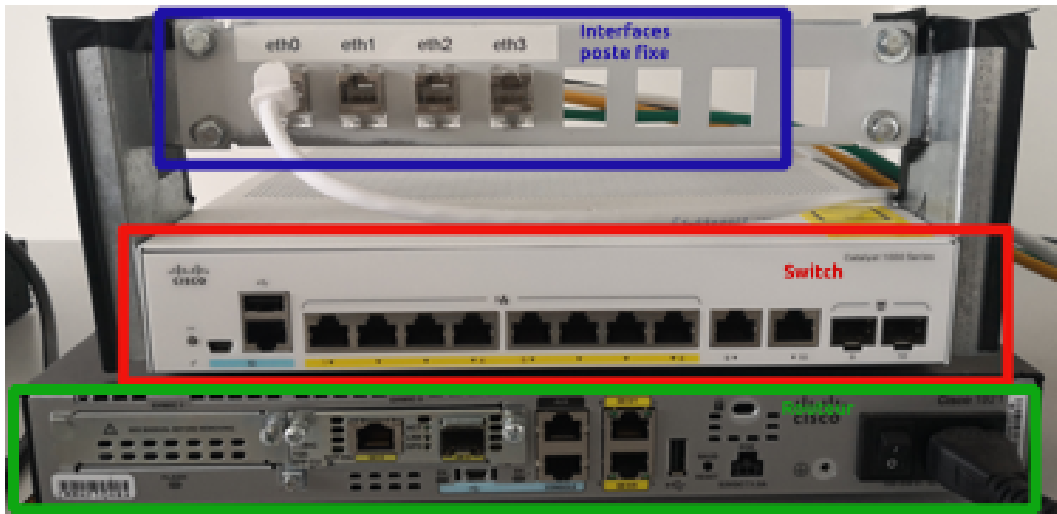


FIGURE 2 – Switch et Routeur

Un **switch** (ou commutateur) est un dispositif permettant de connecter des terminaux **appartenant au même sous-réseau**. Il ne nécessite pas de configuration particulière pour ce type d'usage.

19.14. Parmi les couples d'adresses suivants, lesquels contiennent deux adresses appartenant au même sous-réseau ?

Adresse IP 1/masque	Adresse IP 2/masque	Même sous réseau ?
10.10.1.2/24	172.2.1.13/24	
10.10.1.2/24	10.10.1.1/24	
10.10.1.2/24	10.10.2.1/24	
10.10.1.2/16	10.10.2.1/16	
10.10.1.2/20	10.10.2.1/20	

19.15. Les adresses IP de vos deux VM appartiennent-elles au même sous-réseau ?

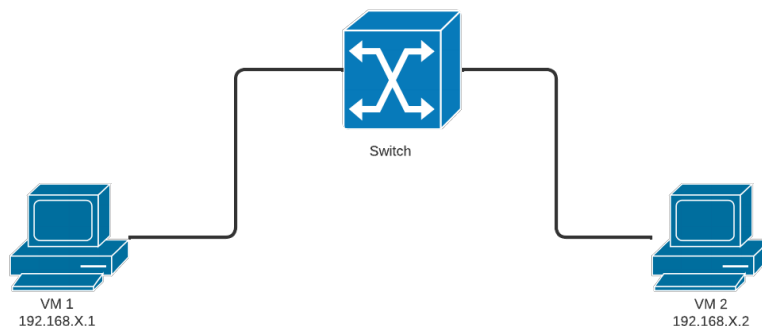


FIGURE 3 – Schéma montage avec switch

19.16.

- a. Brancher le switch.
- b. Relier les interfaces associées aux VM au switch (peu importe les ports utilisés), comme schématisé sur la Figure 3.
- c. Attendre la synchronisation (les LED des ports du switch deviennent vertes une fois la connexion effectuée).
- d. Tester la connectivité entre les machines.

Serveur Web

Il est possible de lancer un serveur web sur une machine avec la commande :

```
systemctl start apache2
```

19.17. Lancez un serveur web sur la VM1. Vérifiez, en ouvrant un navigateur dans la VM1, que vous pouvez bien y accéder à l'adresse 127.0.0.1 (ou localhost).

19.18. Vérifiez, en ouvrant un navigateur dans la VM2, que le serveur web de la VM1 est accessible depuis la VM2 (en rentrant l'adresse de la VM1).

19.19. Quelle est la pile de protocoles utilisée par le serveur web ? Par rapport au ping, quelles informations supplémentaires sont nécessaires pour l'établissement de la connexion ?

Lors du débogage de vos TP, il est souvent utile de faire 2 choses :

- un **ping**, pour vérifier que les machines (ou VM) sont connectées au niveau de la couche internet (IP),
- une requête vers un serveur web, pour vérifier que les machines permettent une connexion au niveau de la couche transport (avec une bonne gestion des ports).

Réseau entre binômes

19.20. À l'aide d'un autre câble, connectez votre switch avec celui du binôme à côté de vous.

19.21. Essayer de faire un ping entre les VM d'un binôme et l'autre. Est-ce que cela fonctionne ? Pourquoi ? Proposez une solution pour les faire communiquer et appliquez-là.

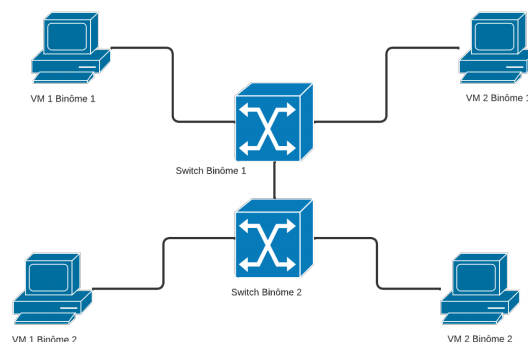


Schéma montage en binôme

Procédure de nettoyage et rangement

19.22. À la fin de votre TP, prévoyez **5 minutes** pour remettre votre espace en ordre :

- Débranchez les câbles et rangez-les derrière votre poste fixe.
- **Ne débranchez pas les câbles reliés aux interfaces du poste fixe !**
- Débranchez le switch.

- Supprimez les VM que vous avez créées.
- Supprimez les documents (compte-rendu, capture d'écran, etc.) que vous avez créés pendant le TP.

6 Découverte salle réseau 2

Exercice 20

L'objectif principal de ce TP est la découverte de la manipulation des routeurs.

Compte rendu Pour ce TP, vous devrez produire un compte-rendu (non-noté pour ce TP). Pour ce faire, utilisez un logiciel (libreOffice, google doc ou autre) **sur le poste fixe !**

Préliminaires

20.1. Mise en place :

- Lancez deux machines virtuelles Linux appelée VM1 et VM2.
- Affectez-leur des adresses IP :
 - ◊ 10.10.X.1/24 pour la VM1,
 - ◊ 30.30.X.1/24 pour la VM2.
- Allumez le routeur.
- Dans votre compte-rendu, expliquez la mise en place des adresses IP (liste de(s) commande(s)), et montrer par une capture d'écran que les IP ont bien été données comme demandées. *Inutile de décrire l'installation de vos VM, précisez seulement à quelle interface du poste fixe vous les avez affectées.*

20.2. On souhaite connecter les VM. Peut-on utiliser uniquement un switch (expliquer pourquoi dans le compte-rendu) ?

Découverte du routeur

Un **routeur** est un dispositif permettant de connecter **plusieurs sous-réseaux distincts**. Il **nécessite** une configuration : contrairement au switch, il va falloir affecter aux interfaces du routeur des adresses IP et les activer pour pouvoir les utiliser.

Il ne faudra JAMAIS débrancher les routeurs ! À la différence des switch, les routeur possèdent un interrupteur qu'on utilisera pour les allumer et les éteindre.

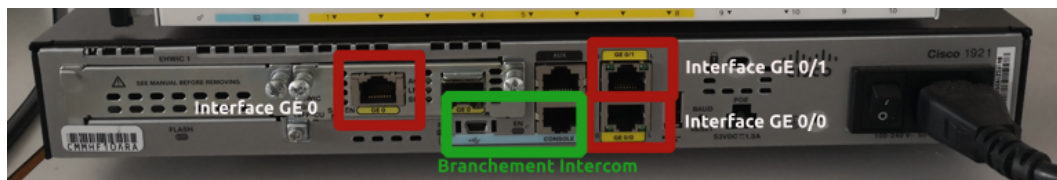


FIGURE 4 – Interface et intercom du routeur

Les routeurs des salles réseaux ESIR-Istic possèdent 3 interfaces (voir Figure 4) qui leur permettent de relier 3 sous-réseaux distincts (**Attention** les interfaces n'ont pas les mêmes noms suivant les salles).

Configuration du routeur

La configuration d'un routeur ne se fait pas directement sur le matériel (à la différence, par exemple, d'un oscilloscope qui possède des boutons physiques ou un écran intégré), il va falloir se connecter au routeur (avec un câble) depuis le poste fixe et modifier sa configuration en utilisant un logiciel (ici via le terminal).

20.3. Reliez le routeur au poste fixe en utilisant le câble intercom.

20.4. Sur le poste fixe, ouvrez un terminal et taper la commande :

```
$ sudo minicom
```

- *Le terminal peut afficher plusieurs informations (appelées log), puis vous rendra la main.*
- *Si vous le faite juste après avoir allumé le routeur, cela peut prendre jusqu'à 2 minutes.*
- *N'hésitez pas à appuyer sur 'Entrée' à certains moments pour voir si vous avez à nouveau la main.*
- ***Si on vous propose de configurer le terminal, REPONDEZ NON !***

Vous devez obtenir une invite de commande du type :

```
Router>
```

Il existe deux modes sur le routeur : utilisateur et super-utilisateur (administrateur). Juste après l'allumage, vous êtes en mode utilisateur et vous ne pourrez faire que de la consultation de configuration. Pour modifier cette configuration, il faut passer en mode super-utilisateur.

20.5. Pour ce faire, tapez la commande :

```
Router> enable
```

Vous êtes alors super-utilisateur et votre invite de commande doit changer en :

```
Router#
```

En général, le passage en mode super-utilisateur nécessite un mot de passe, mais ils ont été désactivé sur les routeurs de cette salle.

Le terminal du routeur fonctionne un peu différemment par rapport à un terminal bash classique :

- il n'y a pas de système de fichiers à proprement parler, les commandes de manipulation de fichiers/dossiers (`cd`, `ls`, `mkdir`) n'existent donc pas.
- comme pour bash, la touche 'Tabulation' permet d'auto-compléter des commandes.
- en bash, un double appui sur 'Tabulation' permettait d'afficher l'ensemble des commandes/fichiers disponibles. Un raccourci similaire existe dans le terminal du routeur : en écrivant ? puis 'Entrée'.

- contrairement à bash, les commandes n'ont pas besoin d'être entrées entièrement pour être exécutées. Exemple :

```
Router# configure terminal
```

peut être abrégée en :

```
Router# conf term
```

Aperçu des commandes classiques Lorsque l'invite de commande affiche `Router#`, cela signifie qu'on se trouve à la 'racine' du routeur. Dans cet espace, on peut notamment utiliser les commandes :

- `ping ADRESSE` : permet de lancer un ping
- `show running-configuration` : permet d'afficher la configuration actuelle du routeur, abrégable en `show run`
- `configure terminal` : permet de se 'déplacer' dans l'espace de configuration du routeur, abrégable en `conf term`

20.6. Affichez la configuration du routeur. Appuyez sur 'Espace' pour faire rapidement défiler l'affichage. Appuyez sur 'q' pour stopper directement l'affichage.

20.7. Sur votre compte-rendu, listez les commandes effectuées jusqu'ici sur le routeur et insérer une capture d'écran de la configuration (au niveau de la partie 'interfaces').

20.8. Entrez dans l'espace de configuration du routeur. Votre invite de commande doit changer en :

```
Router(config)#
```

20.9. Essayez d'afficher à nouveau la configuration. Que se passe-t-il ?

Pour revenir à la racine, il faut 'sortir' de l'espace de configuration en tapant la commande `exit`.

20.10. Revenez à la racine et affichez à nouveau la configuration.

On a souvent besoin d'effectuer des commandes de la racine (ping, affichage de la configuration) et c'est assez pénible de devoir retourner systématiquement à la racine pour les effectuer. Aussi il existe un moyen de spécifier qu'une commande doit être exécutée à la racine : en précédant la commande du mot clé '`do`'.

20.11. Entrez dans l'espace de configuration et taper la commande :

```
Router(config)# do show run
```

Configuration d'une interface Dans la plupart des cas, on cherche à configurer les interfaces du routeur. Pour ce faire il faut, à partir de l'espace de configuration, se 'déplacer' dans l'espace des interfaces que l'on souhaite configurer avec la commande :

```
Router(config)# interface TYPE NUMBER
```

Selon la salle (205 ou 207), TYPE sera FastEthernet et/ou GigabitEthernet et NUMBER, 0/0/0, 0/0/1, 0/0, etc.

Pour savoir quelles interfaces sont disponibles, vous pouvez utiliser les commandes :

```
Router(config)# interface ?
```

ou (par exemple si on choisit TYPE=GigabitEthernet)

```
Router(config)# interface GigabitEthernet ?
```

20.12. Déplacez-vous dans l'espace de configuration d'une des interfaces du routeur, votre invite de commande doit changer en :

```
Router(config-if)#
```

Pour affecter une adresse IP à une interface, on utilise la commande :

```
Router(config-if)# ip address ADDRESS MASK
```

où ADDRESS est l'adresse IP au format W.X.Y.Z, et MASK est le masque du sous-réseau.

Écriture du masque : Attention ! Contrairement à la commande du terminal classique, dans laquelle on spécifie uniquement le nombre de bits correspondant au masque du sous réseau, ici on l'écrit sous la forme (par exemple) :

11111111.11111111.11000000.00000000

- les 1 représentent les bits réservés à l'identification du réseau.
- les 0 représentent les bits réservés à l'identification du terminal dans le réseau.

Et cette écriture doit se faire **en décimal**, dans l'exemple précédent cela donnerait 255.255.192.0.

La plupart du temps, les masques de sous-réseau sont des multiples de 8, ce qui signifie que les masques seront souvent de la forme 255.255.0.0 ou 255.255.255.0.

20.13. Choisissez une adresse IP pour l'interface (elle doit être dans le même sous-réseau que l'une de vos VM).

En général, dans un sous-réseau, on choisit les adresses les plus basses pour les machines et les plus hautes pour les routeurs.

Exemple : dans le réseau 192.168.1.0/24 :

- les machines seront identifiées par 192.168.1.1, 192.168.1.2, ...
- les routeurs par 192.168.1.254, 192.168.1.253, ...

(Rappel : 192.168.1.255 est une adresse réservée).

Important Par défaut, les interfaces sont 'éteintes' (**shutdown**). Il faut donc les activer par la commande :

```
Router(config-if)# no shutdown
```

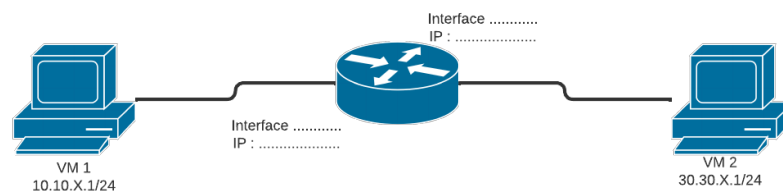
Plus généralement, une commande du type **no COMMAND** annule la commande **COMMAND**. Cela sert également si l'on souhaite supprimer une adresse IP, on utilise alors la commande **no ip address**

20.14. Affichez la configuration du routeur et vérifiez que vous avez bien configuré votre première interface.

20.15. Configurez également la deuxième interface et vérifiez votre configuration.

20.16. Dans votre compte rendu, listez les commandes effectuées (pour une interface), et insérez une capture d'écran de la configuration obtenue.

20.17. À l'aide de deux câbles RJ45, relier les VMs à leur interfaces selon le schéma :



Premier montage avec un routeur

20.18. Sur votre compte-rendu, recopier et compléter le schéma (soit vous le refaites à la main, soit avec des logiciels spécialisés : pour ma part j'utilise lucid.app (en ligne) avec un compte gratuit).

20.19. Effectuez un ping depuis le routeur jusqu'à vos VM. Résolvez les éventuels problèmes, et insérez une capture d'écran dans votre compte-rendu.

20.20. Effectuez un ping depuis vos VM jusqu'aux interfaces correspondantes du routeur (insérez une capture d'écran).

20.21. Effectuez un ping depuis la VM1 jusqu'à l'interface du routeur connectée à la VM2. Que se passe-t-il ? (Expliquez l'erreur sur votre compte-rendu).

Pour résoudre ce problème, il faut spécifier sur les VM que les paquets ayant des destinations inconnues doivent être envoyés à l'adresse du routeur. Cela se fait en spécifiant une 'route par défaut' avec la commande :

```
$ ip route add default via IP_ADDRESS
```

où IP_ADDRESS est l'adresse IP de l'interface correspondante du routeur.

20.22. Définissez une route par défaut sur chacune des vos VM, vérifiez la bonne prise en compte de vos commandes en tapant :

```
$ ip route
```

(insérez une capture d'écran).

20.23. Vérifiez que vos VM peuvent à présent communiquer (insérez une capture d'écran).

Procédure de nettoyage et rangement

20.24. À la fin de votre TP, prévoyez **5 minutes** pour remettre votre espace en ordre :

- Débranchez les câbles et rangez-les derrière votre poste fixe.
- **Ne débranchez pas les câbles reliés aux interfaces du poste fixe !**
- **Supprimer la configuration du routeur** avec la commande :

```
Router# erase startup-config
```

- Éteignez le routeur **sans le débrancher !**
- Supprimez les VM que vous avez créées.
- Enregistrez les documents que vous souhaitez conserver sur votre espace document (ENT → 'Espace documents') ou sur une clé USB.
- Supprimez les documents (compte-rendu, capture d'écran, etc.) que vous avez créés pendant le TP.

7 Gestion multi-routeurs

Exercice 21

L'objectif de ce TP (compte-rendu noté) est de mettre en pratique ce que vous avez appris sur les routeurs et de découvrir comment gérer un réseau multi-routeurs.

Préliminaires

On possède 4 machines ayant comme IP :

- 10.10.10.1/24 pour la machine 1
- 20.20.20.1/24 pour la machine 2
- 30.30.30.1/24 pour la machine 3
- 40.40.40.1/24 pour la machine 4

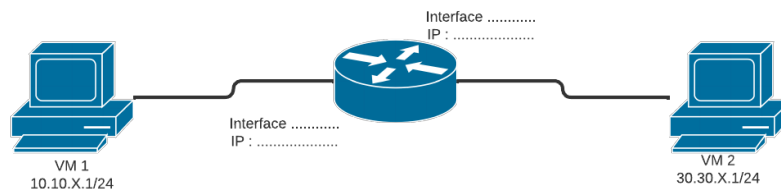
On souhaite les interconnecter en utilisant des routeurs ayant 3 interfaces.

21.1. Proposez une organisation du réseau permettant de les interconnecter. *On cherchera à minimiser le nombre de routeurs.*

Vous en ferez un schéma détaillé sur votre compte-rendu (en précisant les adresses IP de **toutes** les interfaces).

Réalisation d'un montage simple

21.2. Réalisez le montage suivant (X est votre numéro de poste). *Il faut que les 2 VM puissent communiquer.*



Premier montage

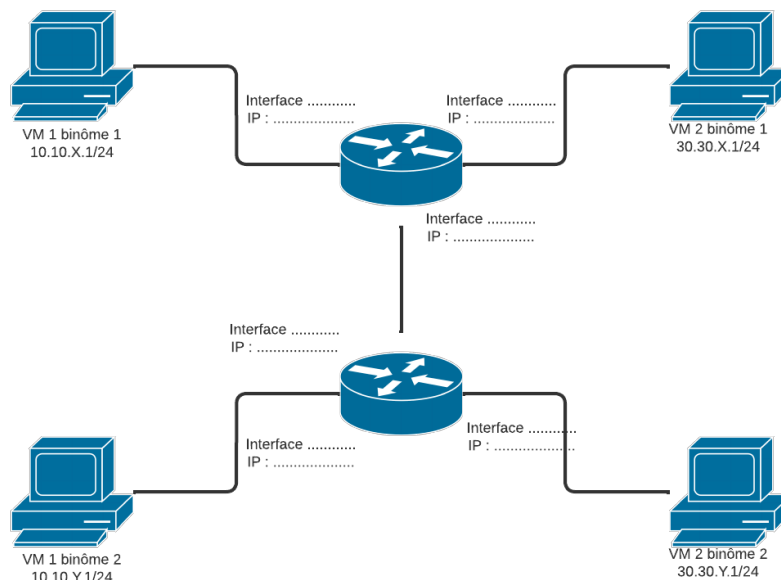
Dans votre compte-rendu :

- vous recopierez le schéma complété avec :
 - ◊ le nom des interfaces utilisées (pour le routeur)
 - ◊ les adresses IP de **toutes** les interfaces
- vous listerez les commandes que vous avez effectuées, en précisant à chaque fois que c'est possible les tests que vous avez effectués pour vous assurer que la commande a bien été prise en compte (en insérant des captures d'écran).

Liaison entre deux binômes : routage statique

À présent, on souhaite faire communiquer votre réseau avec le réseau du binôme voisin.

21.3. À l'aide d'un câble RJ45, reliez les interfaces restantes des 2 routeurs et configurez les de manière à ce qu'ils puissent communiquer (détaillez votre procédure dans votre compte-rendu).



Deuxième montage

21.4. Depuis votre routeur, effectuez un ping vers le routeur voisin. Cela fonctionne-t-il ? (*insérez une copie d'écran dans votre compte-rendu*)

21.5. Toujours depuis votre routeur, effectuez un ping vers l'une des VM du binôme voisin. Cela fonctionne-t-il ? (*insérez une copie d'écran dans votre compte-rendu*)

La raison pour laquelle ce dernier ping ne fonctionne pas est que votre routeur ne 'connaît' que les 3 sous-réseaux auxquels il appartient : les réseaux 30.30.Y.0 et 10.10.Y.0 lui sont inconnus.

Pour lui permettre de communiquer avec des sous-réseaux 'lointains', il faut configurer le routeur en lui indiquant vers quelle adresse directement accessible transmettre quels paquets.

Cela se fait par la commande (depuis l'espace de configuration du routeur) :

```
Router(config)# ip route DEST_ADDRESS DEST_MASK TARGET_ADD
```

où :

- DEST_ADDRESS DEST_MASK est l'adresse/masque du réseau que l'on cherche atteindre (par exemple 10.10.Y.0 255.255.255.0)
- TARGET_ADDRESS est l'adresse du terminal **directement connecté à notre routeur** vers lequel on enverra ces paquets.

21.6. Configurez votre routeur pour pouvoir atteindre les sous-réseaux de votre binôme voisin.

21.7. Testez à nouveau un ping depuis votre routeur. (*insérez une copie d'écran dans votre compte-rendu*)

21.8. Depuis l'une de vos VM, essayer d'envoyer un ping vers le routeur de votre binôme. *Si cela ne fonctionne pas, c'est peut-être que vos voisins n'ont pas encore défini de route au niveau de leur routeur. (insérez une copie d'écran dans votre compte-rendu)*

En pratique le routage ne se fait pas de manière statique : les routeurs échangent des informations entre eux de manière automatique pour mettre à jour les réseaux accessibles, les meilleures routes, etc.

Cela permet au réseau d'être plus résilient aux pannes, coupures, etc. Mais cela est plus difficilement observable et simulable en salle de TP.

Question subsidiaire

21.9. Si on souhaite mettre en réseau $2n$ machines appartenant toutes à des sous-réseaux différents, en n'utilisant que des switches et des routeurs à 3 interfaces, combien faudra-t-il de routeurs (au minimum)? *(proposez un schéma dans votre compte-rendu)*

Procédure de nettoyage et rangement

21.10. À la fin de votre TP, prévoyez **5 minutes** pour remettre votre espace en ordre :

- Débranchez les câbles et rangez-les derrière votre poste fixe.
- **Ne débranchez pas les câbles reliés aux interfaces du poste fixe !**
- **Supprimer la configuration du routeur** avec la commande :

```
Router# erase startup-config
```

- Éteignez le routeur **sans le débrancher !**
- Supprimez les VM que vous avez créées.
- Enregistrez les documents que vous souhaitez conserver sur votre espace document (ENT → 'Espace documents') ou sur une clé USB.
- Supprimez les documents (compte-rendu, capture d'écran, etc.) que vous avez créés pendant le TP.

8 NAT

Exercice 22

L'objectif de ce TP⁴ est de découvrir un procédé permettant notamment de démultiplier les adresses IP : le NAT (*Network Address Translation*).

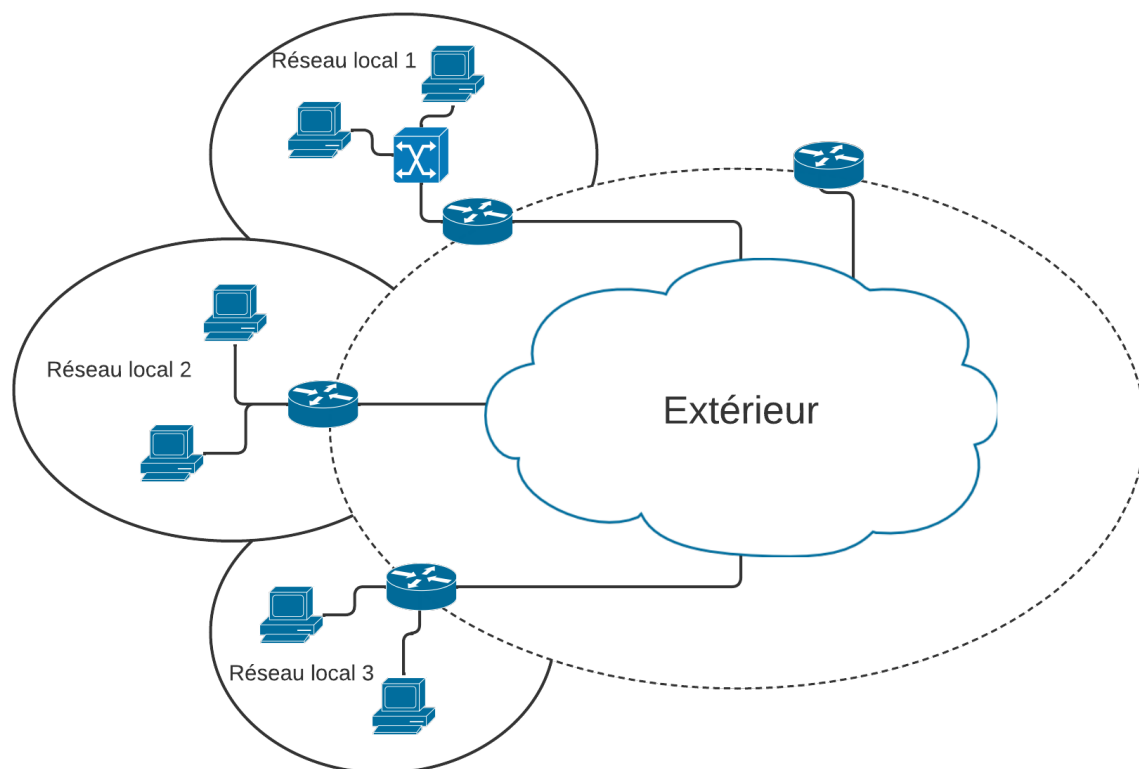
Contexte

22.1. Rappelez le nombre d'adresses IP (v4) possibles.

Dans les années 1990, les adresses IPv4 ont commencé à manquer. Pour palier à ce problème, plusieurs solutions ont été mises en place, dont notamment :

- la mise d'adresses IPv6, sur 128 bits (utilisées pour la première fois en 1999)
- l'utilisation de translation d'adresse (NAT), qui permet d'affecter la même adresse IP à plusieurs terminaux dans un même réseau (utilisé à partir de 2000).

Principe de fonctionnement Pour ce TP, on considère qu'internet peut être vu comme une agrégation de plusieurs réseaux locaux. Chaque réseau local n'est connecté à internet que par un routeur (typiquement les box des particuliers).

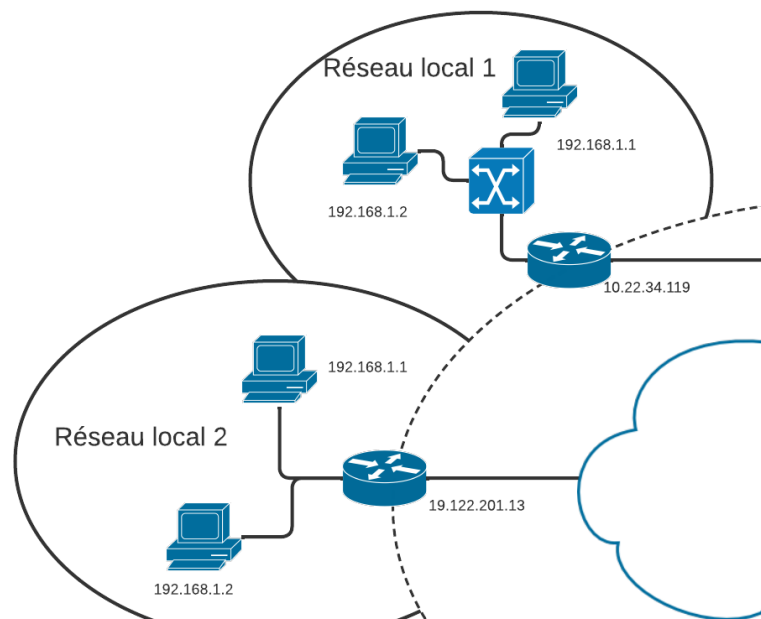


Internet comme agrégation de réseaux locaux

Dans cette configuration, les terminaux internes des différents réseaux locaux peuvent partager les mêmes adresses IP, car on peut distinguer la manière d'y accéder (les routeurs ayant des adresses distinctes).

4. inspiré du TP d'ESIR1-RES de A. Baire, Y. Hadjadj-Aoul, G. Guette

On peut par exemple avoir :



- Les machines du réseau local 2 ont des adresses IP déjà attribuées dans le réseau local 1.
- Les routeurs (box) qui connectent les réseaux à l'extérieur ont des adresse IP distinctes.
- Ainsi, dans l'idée d'une connexion à une machine de l'extérieur, il est possible de distinguer la machine avec l'IP 192.168.1.1 derrière le routeur 10.22.34.119 de la machine avec l'IP 192.168.1.1 derrière le routeur 19.122.201.13.

Le principe du NAT est implémenté au niveau du routeur. Prenons la situation suivante : la machine (appelons-la PC1) du réseau local 1 dont l'IP est 192.168.1.1 cherche à communiquer avec l'extérieur (exemple : consulter un site web dont le serveur a pour adresse 18.18.18.18) :

1. PC1 prépare un paquet IP avec :
 - 18.18.18.18 comme adresse destination,
 - 192.168.1.1 comme adresse source.
 2. Il transmet le paquet au routeur (appelons-le R1, d'adresse IP 10.22.34.119).
 3. R1 **modifie** le paquet IP avec :
 - 18.18.18.18 comme adresse destination,
 - comme adresse source.
 4. R1 envoie le paquet dans le réseau.
 5. Le réseau répond à R1 (nouveau destinataire) un paquet IP avec :
 - 10.22.34.119 comme adresse destination,
 - 18.18.18.18 comme adresse source.
 6. R1 **modifie** le paquet IP reçu avec :
 - 192.168.1.1 comme adresse destination,
 - 18.18.18.18 comme adresse source.
 7. R1 transmet le paquet modifié à PC1.
- Du point de vue de PC1, tout est transparent :

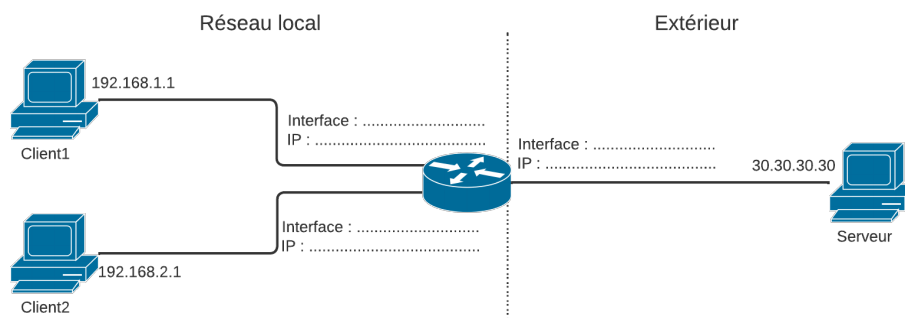
- ◊ il a envoyé un paquet avec son adresse comme source et 18.18.18.18 en destination,
 - ◊ et a reçu un paquet réponse avec les adresses inversées : comme s'il était sur un réseau 'classique'.
 - Du point de vu du serveur web, PC1 n'existe pas et seul le routeur existe :
 - ◊ il a reçu un paquet de l'adresse 10.22.34.119,
 - ◊ et a renvoyé sa réponse vers la même adresse.
- On dira que 10.22.34.119 est **l'adresse publique** du réseau local 1 (l'adresse visible par le monde extérieur) et elle est la même pour tous les terminaux internes de ce réseau.

Modélisation

Pour modéliser un réseau avec besoin de NAT, nous utiliserons 3 VM :

- 2 VM représentant le réseau local, nommées Client1 et Client2.
- 1 VM représentant le monde extérieur, nommé Serveur

Ces VM seront câblés selon le schéma suivant :



22.2. Réalisez le montage ci-dessus. Vous préciserez dans votre compte-rendu la liste des commandes effectuées pour le réaliser ainsi que les tests effectués pour vérifier la bonne connectivité.

Attention! Il ne faut pas mettre de route par défaut sur le serveur!!
Il faut imaginer que le serveur représente l'ensemble d'internet, il ne va donc pas envoyer les paquets dont il ne connaît pas la destination par défaut vers votre box à vous.

22.3. Lancer un ping depuis le Client1 vers le Serveur. Cela fonctionne-t-il? Expliquez dans votre compte-rendu pourquoi.

Translation d'adresse statique

Nous allons à présent configurer le routeur pour lui permettre de faire de la translation d'adresse.

Dans un premier temps, il faut lui signifier quelles interfaces correspondent à l'*extérieur* (**outside**) et quelles interfaces font parties du réseau local (**inside**). Pour ce faire, on utilise la commande (dans l'espace de configuration de chaque interface) :

```
Router(config-if)# ip nat outside
```

ou

```
Router(config-if)# ip nat inside
```

22.4. Pour chacune de vos interfaces, définissez si elle est dans la partie **inside** ou **outside** du NAT (insérez une capture d'écran de la configuration du routeur dans votre compte-rendu).

Pour que le NAT fonctionne, il faut également spécifier les règles de redirections. Dans un premier temps, une translation **statique** peut être mise en place (depuis l'espace de configuration du routeur) à l'aide de la commande :

```
Router(config)# ip nat inside source static  
PRIVATE_IP PUBLIC_IP
```

où :

- PRIVATE_IP est l'adresse de la machine du réseau local que l'on souhaite nater,
- PUBLIC_IP est l'adresse publique du routeur.

22.5. Mettez en place une translation statique pour le Client1, et testez la connexion entre le Client1 et le Serveur (avec un ping et en démarrant un serveur web sur le Serveur). *Insérez une capture d'écran dans votre compte-rendu.*

Vous pouvez vérifier que la règle a bien été mise en place en tapant la commande (depuis la racine) :

```
Routeur# show ip nat translations
```

22.6. Essayez d'accéder au Serveur depuis Client2. Que se passe-t-il ?

22.7. Essayez de mettre en place une deuxième translation statique pour Client2. Que se passe-t-il ?

Translation d'adresse et de port

Le problème de la translation statique est qu'on ne peut définir qu'une seule règle de translation.

22.8. Supprimer la règle de la translation statique que vous aviez mis en place. (*Insérez une capture d'écran de la liste des règles mises en place*)

Pour pallier à ce problème, on va utiliser une *translation d'adresse et de port*. Pour ce faire, il faut :

1. définir une 'liste d'accès' des adresses IP ayant le droit de se faire traduire, à l'aide de la commande :

```
Router(config)# access-list NUM permit  
PRIVATE_NETWORK ANTI_MASK
```

où :

- NUM est un numéro à choisir entre 1 et 99
- PRIVATE_NETWORK est le sous-réseau dont les adresses pourront se faire nater (Ex : 192.168.1.0)
- ANTI_MASK est le complémentaire du masque du sous-réseau. (Ex : l'anti-masque de 255.255.255.0 est 0.0.0.255)
- Plusieurs plages d'adresses peuvent être enregistrées dans la même liste, il suffit de répéter la commande avec le même numéro de liste.

2. enregistrer et configurer la translation :

```
Router(config)# ip nat inside source list NUM  
                interface INTERFACE_NAME overload
```

où :

- NUM est le numéro de la liste précédemment définie
- INTERFACE_NAME est le nom d'une interface **publique** du routeur.

22.9. Définissez une liste d'accès pour les adresses du sous-réseau de Client1 et enregistrez-la. Vérifiez l'accès au Serveur depuis Client1 (*listez les commandes entrées et insérer des captures d'écran dans votre compte-rendu, avec notamment la configuration des liste de translation*).

22.10. Essayez depuis votre Client2. Cela fonctionne-t-il ?

22.11. Réglez le problème et vérifier la connexion.

Simulation d'un serveur en réseau local

Dans cette partie, on va supposer cette fois-ci que l'on cherche à accéder **depuis** l'extérieur à un service dans le réseau local. Par exemple, on va supposer que Client1 est un serveur web et que Serveur souhaite y accéder.

Conceptuellement, cela suppose de considérer que Client1 va avoir un rôle de serveur et Serveur un rôle de client. En revanche, les branchements et la configuration du routeur vont rester les mêmes.

22.12. Lancez un serveur web sur Client1. À quelle adresse peut-on y accéder depuis Client2 ? Est-ce que ce sera la même adresse pour y accéder depuis Serveur ? La problématique ici est que Serveur ne connaît que l'adresse publique du routeur du sous-réseau de Client1, il faut donc configurer le routeur en spécifiant que si une requête HTTP est effectuée, elle doit être redirigée vers Client1.

22.13. Rappelez quel port est habituellement utilisé pour les requêtes HTTP.

Le routeur ne va pas désencapsuler les paquets jusqu'à la couche application (pour détecter le protocole HTTP), en revanche il peut détecter le protocole de la couche transport et le port utilisé.

Pour rediriger des paquets utilisant le protocole TCP vers une machine du réseau privé, on utilise la commande :

```
Router(config)# ip nat inside source static tcp  
                PRIVATE_IP PRIVATE_PORT PUBLIC_IP PUBLIC_PORT
```

où :

- PUBLIC_IP, PUBLIC_PORT est le couple (adresse IP, port) publique sur lequel récupérer les paquets reçus.
- PRIVATE_IP, PRIVATE_PORT est le couple (adresse IP, port) privée vers lequel rediriger les paquets reçus.

22.14. Mettez en place la redirection vers le Client1 depuis l'adresse publique du routeur pour traiter les requêtes HTTP.

Vérifiez que vous pouvez vous connecter au serveur web du Client1 depuis le Serveur.
(listez les commandes entrées et insérer des captures d'écran dans votre compte-rendu)

Question subsidiaire

22.15. En supposant qu'on lance un serveur Web également sur Client2, comment serait-il possible de pouvoir y accéder depuis Serveur ?

Procédure de nettoyage et rangement

22.16. À la fin de votre TP, prévoyez **5 minutes** pour remettre votre espace en ordre :

- Débranchez les câbles et rangez-les derrière votre poste fixe.
- **Ne débranchez pas les câbles reliés aux interfaces du poste fixe !**
- **Supprimer la configuration du routeur** avec la commande :

```
Router# erase startup-config
```

- Éteignez le routeur **sans le débrancher !**
- Supprimez les VM que vous avez créées.
- Enregistrez les documents que vous souhaitez conserver sur votre espace document (ENT → 'Espace documents') ou sur une clé USB.
- Supprimez les documents (compte-rendu, capture d'écran, etc.) que vous avez créés pendant le TP.