# ENVIRONNEMENT DE TRAVAIL, OUTILS ET PREMIERS PROGRAMMES [1]

## Rendu de TP

Les réponses aux exercices sont à rendre sur votre dépôt github de nom « tp-prenom » (https://github.com/uns-iut-info/tp-prenom.git), en respectant les consignes suivantes :

- ☐ tous les fichiers sont placés sous le répertoire `A311/TP_ARG_ENV_ID` (en majuscule), vous ne devez pas inclure les exécutables ;
- ☐ **dans le répertoire** `A311/TP_ARG_ENV_ID`**, vous devez créer un fichier vide (en respectant la typographie sans accents) :**
  `TP_ARG_ENV_ID_Prenom1_NOM1_et_Prenom2_NOM2`
- ☐ un fichier `README.md` qui décrit votre avancement : exercices terminés, en cours, difficultés éventuelles rencontrées avec le poste de travail, ...
- ☐ si nécessaire, fournir un document texte de nom `exercice_num.md` avec les réponses de l'exercice numéro $num$ ;
- ☐ les fichiers sources C **commentés avec votre/vos noms en en-tête** doivent utiliser le codage UTF-8 et respecter les noms de l'énoncé ;
- ☐ fournir un fichier `Makefile` qui permet de compiler sans erreur tous les programmes (`make all`) et supprimer les exécutables (`make clean; make mrproper`);
- ☐ chaque programme doit proposer dans la fonction `main()` des tests en mode silencieux (aucune entrée/sortie interactive).

---

## 1 argv et getenv()

Programmez les exemples du cours et testez les.

- `echoargenv.c` (diapositive 21, coté Unix seulement)
- `echoargenviron.c` (diapositive 23)
- `exemple_environ_2.c` (diapositive 24)

On rappelle que pour compiler un programme C simple en ligne de commande, il suffit de lancer :

```
$ gcc -std=gnu99 -Wall -o echoargenv echoargenv.c
```

On détaillera les options du compilateur `gcc` dans un prochain exercice. Pour le premier exemple on doit fournir un script bash `teste_echoargenv.sh` qui lance au moins les exécutions suivantes :

```
echoargenv
echoargenv PATH HOME PWD
echoargenv TOTO
export TOTO="mon test de TOTO"
echoargenv TOTO
```

1. voir Cours_INTRO.pdf

## 2 Makefile

En vous inspirant du fichier `Makefile` de la diapositive 26, écrivez le `Makefile` qui permet de compiler les trois exemples précédents (fichier à inclure dans le même répertoire que les sources).

**Avant de passer à la suite, sauvegardez vos fichiers sur le dépôt github.**

Dans ce répertoire, testez les commandes `make, touch *, make clean` et `make all`.

## 3 Appels systèmes

Commentez l'affichage produit par l'éxécution de la commande `strace` sur l'exécutable `echoargenv` (voir diapos 16).

## 4 myid

Dans une console (terminal), testez la commande bash `id` : quel est le résultat ? Comparez ce résultat avec le contenu du fichier `/etc/passwd` (voir aussi `/etc/nsswitch.conf` et les explications de l'enseignant).

Écrivez le programme C `myid.c` qui imite cette commande en utilisant les appels systèmes `getuid()` et `getgid()`. Voir le manuel avec la commande `man`. Pour le nom de l'utilisateur, voir `getpwnam()` et sa famille (ie `getpwuid)`.

## Annexes pour les futurs TP (version à compléter)

### Configuration git

Configurez le gestionnaire de version `git` en suivant les commandes au tableau :

```
gen-gitconfig.sh (script disponible sur les postes du département INFO)
git clone https://github.com/uns-iut-info/tp-prenom.git  tp-prenom
cd tp-prenom
mkdir -p A311/TP_ARG_ENV_ID
touch A311/TP_ARG_ENV_ID/README.md
git add .
git commit -m "premier commit"
git push origin master

git pull

git-rm
```

Attention, on ne voit pas les répertoires vides.

### Vagrant

Suivre les instructions de :
`https://wiki.unice.fr/display/IntraSIDI/VAGRANT02`
pour installer une machine virtuelle qui vous permettra de réaliser les TP sur votre portable Windows ou Mac OS (pour les IPC non supportées par cet OS).

### Editeur / IDE

Vous pouvez utiliser l'éditeur ou IDE de votre choix à condition qu'il puisse éditer les fichiers sources avec un codage UTF-8. Pour la présentation des sources, soit votre éditeur/IDE vous propose un menu de type Format/Indent, soit vous devez utiliser la commande `indent`.