# Versioning

## GIT - GitLab

H. Djerroud

LIASD - Université Paris 8

# Support de cours

Compte UFR : première connexion

- Login : [première lettre du prénom][Nom] en minuscules
- Mot de passe : Numéro d'étudiant

Moodle

- Nom Cours : Logiciels Libres
- Clé moodle : LOGLIB
- Cours GIT (temporaire)

# Plan de cours

- Introduction aux systèmes de Versioning
- Introduction aux méthodes de développement
- Git
- GitLab

# Outils de versioning à quoi ça sert ?

- Faire des sauvegardes en cours de développement
- Revenir en arrière en cas de besoin
- Travailler à plusieurs sur le même code
- Garder une trace des actions effectuées
- Travailler sur plusieurs fonctionnalités en même temps
- Une façon d'organiser le travail

Plus loin

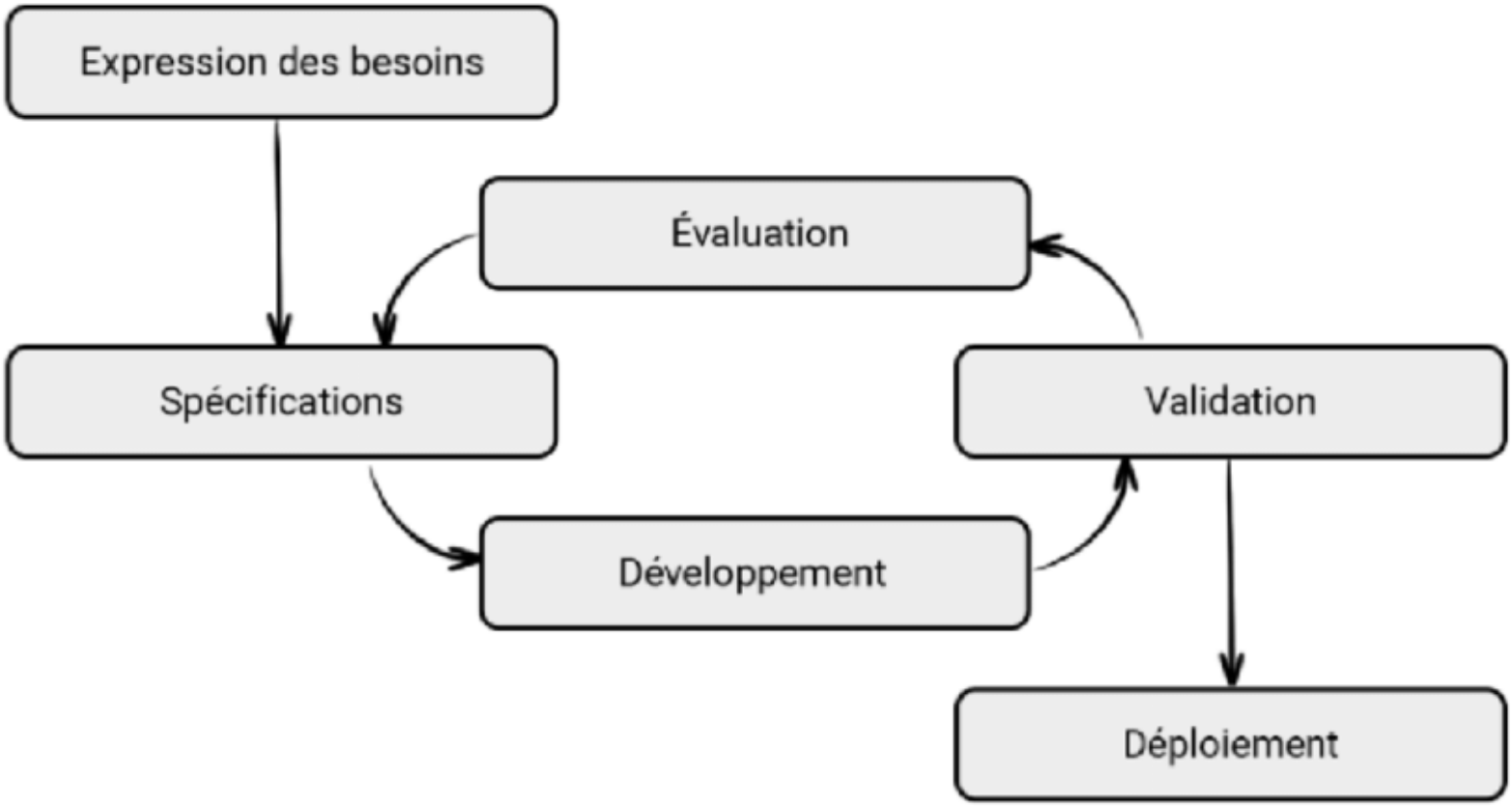
- Tester automatiquement le code source lorsque un développeur rajoute du code
- Re-générer l'application lorsque un développeur rajoute du code

# Historique

- CVS (Concurrent Versions System)
- SVN (Subversion)
- Mercurial
- Monotone (MTN)
- Git

# Méthode itérative et incrémentale et système de versioning

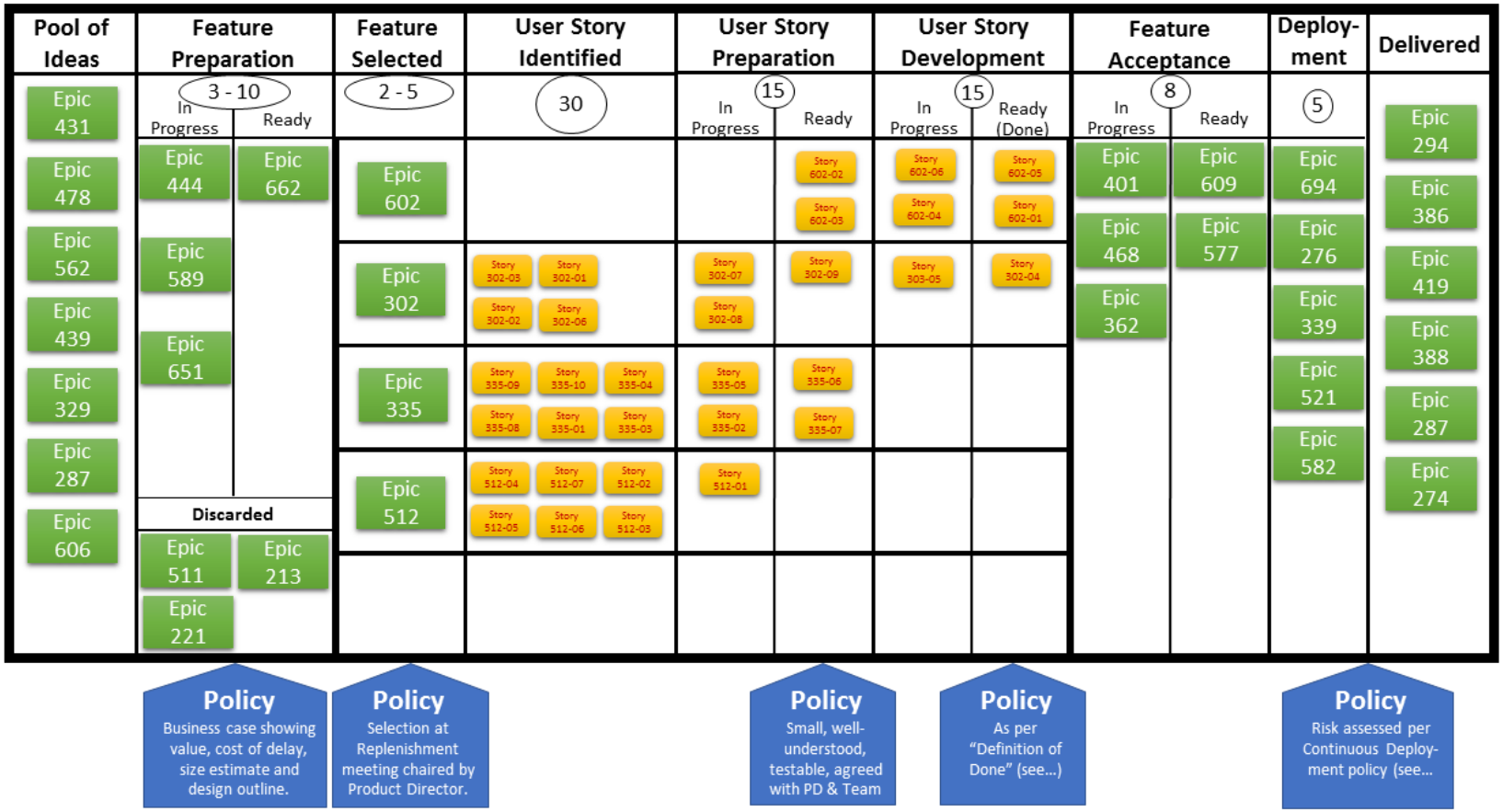
- Cycle itératif et incrémental



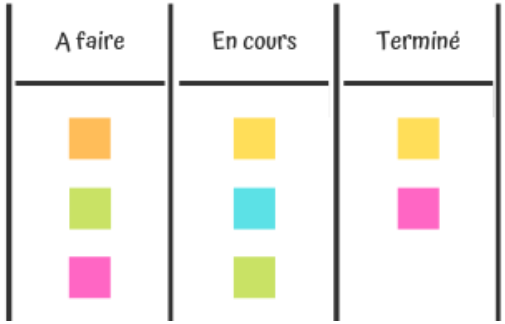
# Lexique dans un framework agile

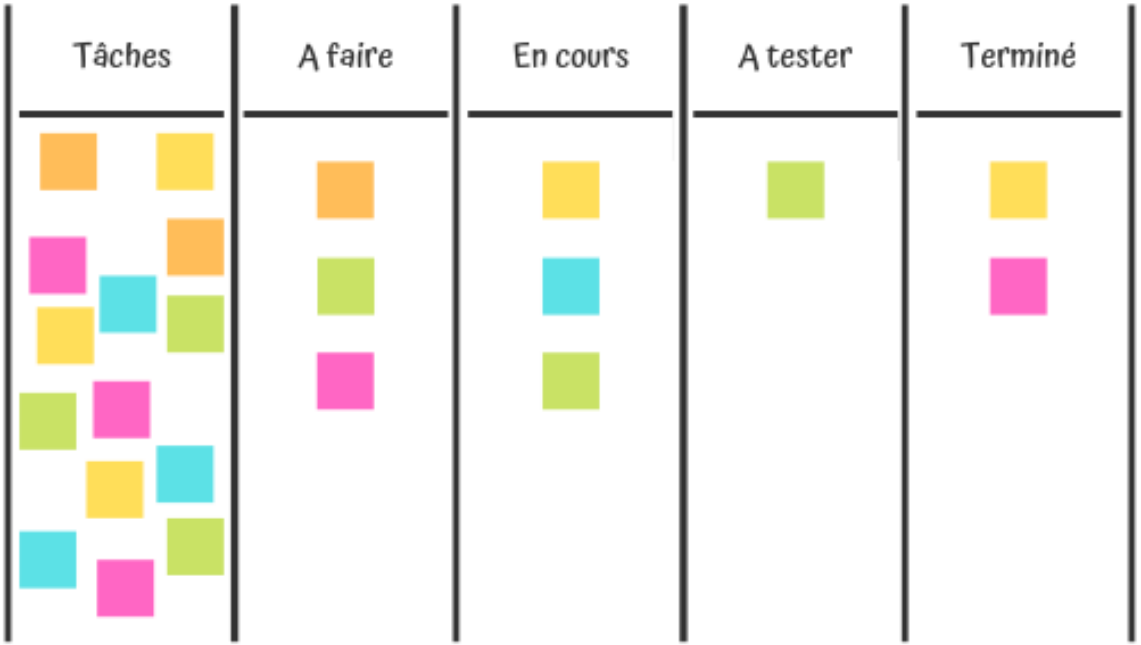
- Features (fonctionnalités)
- User story : explication d'une fonctionnalité logicielle décrite du point de vue de l'utilisateur final
- Epic (User Story non affinée)
- Backlog : une liste de story priorisées
- Kanban

# Kanban


Pool of Ideas
Feature Preparation
3 - 10
In Progress
Ready
Feature Selected
2 - 5
User Story Identified
30
User Story Preparation
15
In Progress
Ready
User Story Development
15
In Progress
Ready (Done)
Feature Acceptance
8
In Progress
Ready
Deploy-ment
5
Delivered
Epic 431
Epic 478
Epic 562
Epic 439
Epic 329
Epic 287
Epic 606
Epic 444
Epic 589
Epic 651
Epic 662
Discarded
Epic 511
Epic 213
Epic 221
Epic 602
Epic 302
Epic 335
Epic 512
Story 602-02
Story 602-03
Story 602-06
Story 602-04
Story 602-05
Story 602-01
Story 302-03
Story 302-01
Story 302-02
Story 302-06
Story 302-07
Story 302-08
Story 302-09
Story 303-05
Story 302-04
Story 335-09
Story 335-10
Story 335-04
Story 335-08
Story 335-01
Story 335-03
Story 335-05
Story 335-02
Story 335-06
Story 335-07
Story 512-04
Story 512-07
Story 512-02
Story 512-05
Story 512-06
Story 512-03
Story 512-01
Epic 401
Epic 468
Epic 362
Epic 609
Epic 577
Epic 694
Epic 276
Epic 339
Epic 521
Epic 582
Epic 294
Epic 386
Epic 419
Epic 388
Epic 287
Epic 274
Policy
Business case showing value, cost of delay, size estimate and design outline.
Policy
Selection at Replenishment meeting chaired by Product Director.
Policy
Small, well-understood, testable, agreed with PD & Team
Policy
As per "Definition of Done" (see...)
Policy
Risk assessed per Continuous Deploy-ment policy (see...

# Kanban peut prendre plusieurs formes



# WeKan : Demo

Depuis les postes fixes : Connectez vous au :

`https://wekan-pedago.bocal.cs.univ-paris8.fr`

- Méthode d'authentification : LDAP
- Login / password : identifiants de votre compte UFR

# GIT : Configuration

- Configuration globale (~/.gitconfig) :

```
$git config --global user.name "Prénom Nom"
$git config --global user.email "monemail@univ-paris8.fr"
```

- Configuration locale (.git/config) :

```
$git config user.name "Prénom Nom"
$git config user.email "monemail@univ-paris8.fr"
```

# GIT : Créer un dépot

- From scratch :

```
$git init
```

- Depuis un dépôt existant :

```
$git clone https://github.com/hdd-robot/fg_saitek.git
```

OU

```
$git clone git@github.com:hdd-robot/fg_saitek.git
```

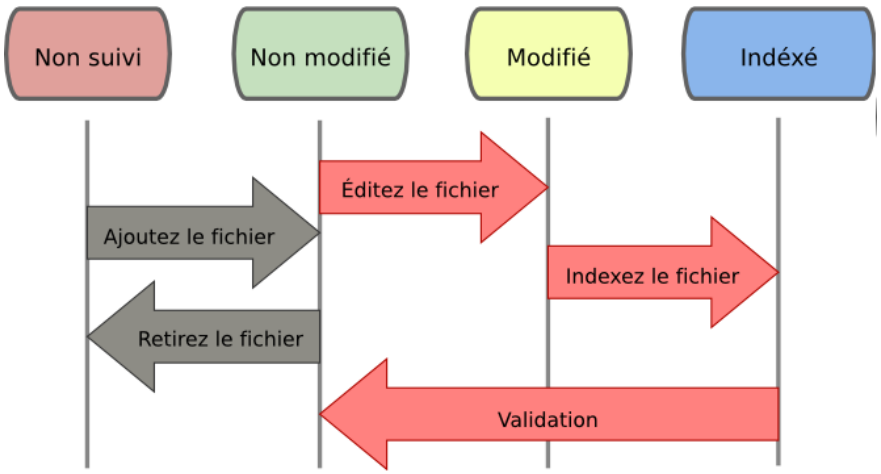
# Git : Cycle de vie d'un fichier

**Non suivi :** fichier qui n'est pas ou plus géré par Git

**Non modifié :** fichier sauvegardé dans la version courante dans la base de données du dépôt

**Modifié :** fichier ayant subi des modifications depuis la dernière fois qu'il a été soumis

**Indexé :** fichier modifié, sauf qu'il sera pris en compte lors de la prochaine soumission (commit)



# Git : Vérifier l'état des fichiers

```
$git status
```

La commande git status affiche l'état du répertoire de travail et de la zone de transit.

# Git : Indexer un fichier

```
$git add NOM_FICHIER

$git add -A     ou    $git add .
```

Indexer l'ajout ou les modifications d'un fichier avant le commit des modifications

# Git : Enregistrer les modifications (COMMIT)

```
$git commit -m "Mon Message de commit"

$git commit  # ouverture automatique d'un editeur
```

Indexer l'ajout ou les modifications d'un fichier avant le commit des modifications

# Git : Visualiser l'historique des validations

```
$git log
```

Par défaut, git log liste en ordre chronologique inversé les commits réalisés.

```
$git log --oneline
```

Affichage sur une seule ligne

```
$git log --graph --decorate --oneline
```

Un affichage plus complet avec les branches

# Git : Visualiser une ancienne version

```
$git log --oneline
```

Visualiser la liste des commits

```
$git checkout <num_commit>
```

Revenir en mode lecture seul dans ancienne version (commit num_commit)

```
$git checkout <master>
```

Revenir à la version courante

# Git : Utilisation d'un dépôt distant

`$git remote show` ou `$git remote -v`

Voir les nom des dépôts distant. Le dépôt par défaut depui un clone se nome `origin`

```
$git remote add nom_remot
   git@gitlab.bocal.cs.univ-paris8.fr:username/nom_depot
```

Ajouter un dépot distant

# Git : Enregistrer des modifications sur un dépôt

```
$git push origin master
```

La commande `push` permet d'envoyer tout les `commits` effectués qui se trouvent dans le répertoire Git/dépôt (HEAD) de la copie du dépôt local vers le dépôt distant.

# Git : Récupérer des modifications d'un dépôt

```
$git pull
```

La commande *pull* permet de mettre à jour le dépôt local avec les dernières validations (modifications des fichiers).

# Git : Créer des branches

```
$git branch feature_x
```

Créer une nouvelle branche nommée `feature_x`

# Git : Basculer vers une branche existante

```
$git checkout feature_x
```

Déplace le pointeur (HEAD) vers la branche `feature_x`. Tous les commits sont effectués sur la branche courante.

```
$git checkout master
```

Retourner sur la branche principale `master`

# Git : Supprimer des branches

```
$git branch  -d feature_x
```

Supprimer la branche `feature_x`

# Git : Fusionner des branches

```
$git merge <branche>
```

Fusion une autre branche avec la branche active.

**Attention**

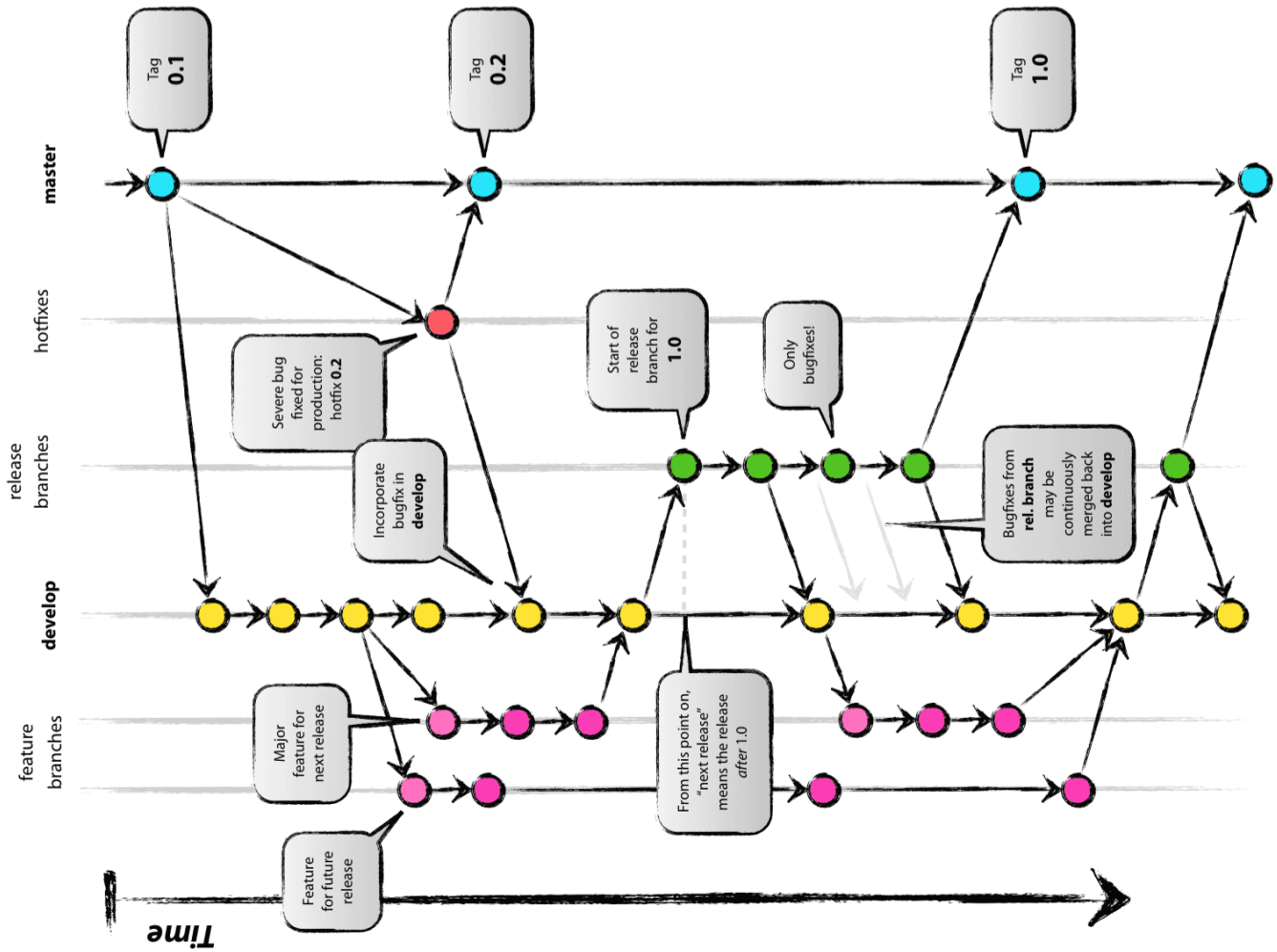
Il est possible qu'il y ait des conflits à résoudre lors d'un merge

# Git : Résoudre les conflits de fusion

Lorsque on modifie différemment la même partie du même fichier dans les deux branches qu'on souhaite les fusionner, Git ne sera pas capable de réaliser proprement la fusion

- Aucun `commit` de fusion n'est créé et le processus est mis en pause.
- Mes conflits doivent être modifiés manuellement en éditant les fichiers indiqués par git :
  - Faire `git status` qui donne les fichiers n'ayant pas pu être fusionnés (listés en tant que *"unmerged"*).
  - Marquer les conflits comme résolus en faisant la commande `git add<fichier>` ou `git commit -a`
  - Après résolution de tous les conflits, soumettre les modifications sous forme d'objet `"commit"` de fusion avec `git commit -m "Mon premier commit"` ou `git push` et terminer ainsi le processus de fusion

# Workflow avec git



# Git Lab : Demo

Depuis les postes fixes : Connectez vous au :

https://gitlab.bocal.cs.univ-paris8.fr

- Login / password : identifiants de votre compte UFR