

Jenkins utilisation: Cahiers de TPs

TP1



Objectifs

- Créer son premier job ;
- Lancer un job;
- Consulter le log d'un build ;



Sujet

- Créer dans Jenkins un job de type Freestyle ;
- Ajouter le code suivant dans le partie build :

echo "Formation ADEO"

• Lancer le job manuellement et consulter le log



Correction

Créer le job en cliquant sur Nouveau item, sélectionner le premier type Construire un projet free-style :

Configurer le job comme suit : dans la partie build, sélectionner dans la Isite Exécutere un script shell :



Résultat attendu dans la console :



TP2

Objectifs

• créer un secret Git

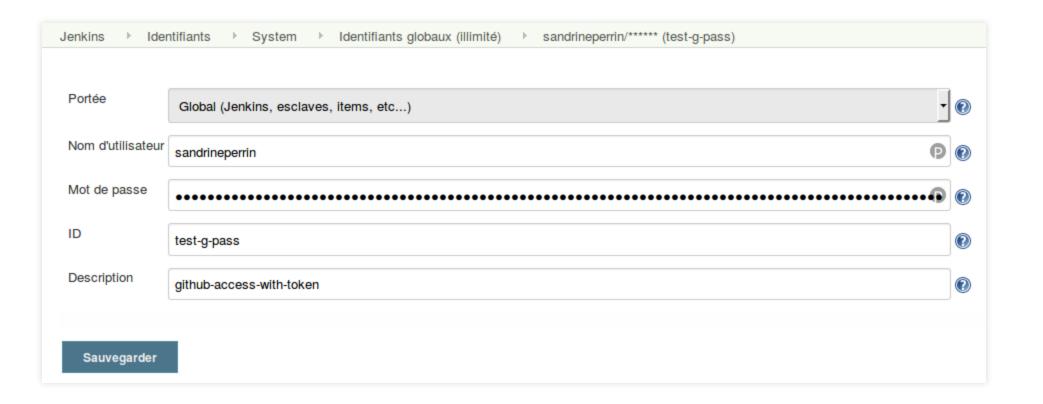


Sujet

- Créer un secret Git global de type utilisateur/mot de passe
- Renseigner:
 - Utilisateur : nom d'utilisateur GitHub
 - Mot de passe : Token GitHub



Correction



TP3



Objectifs

• Créer un job pipeline

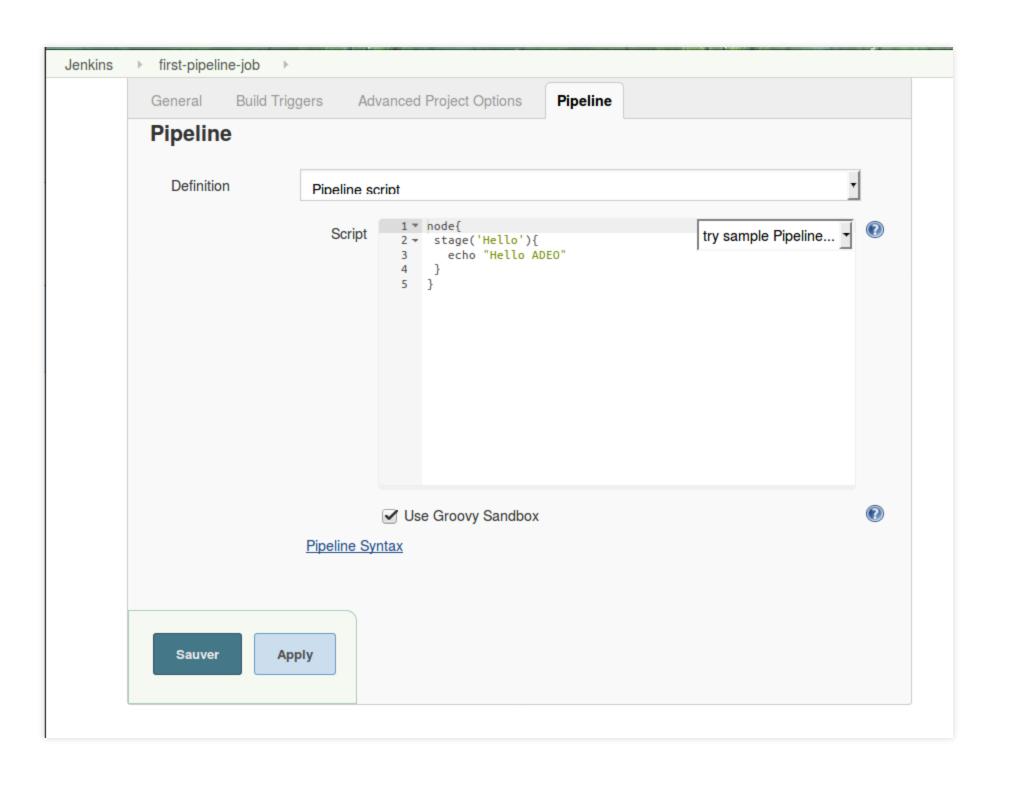


Sujet

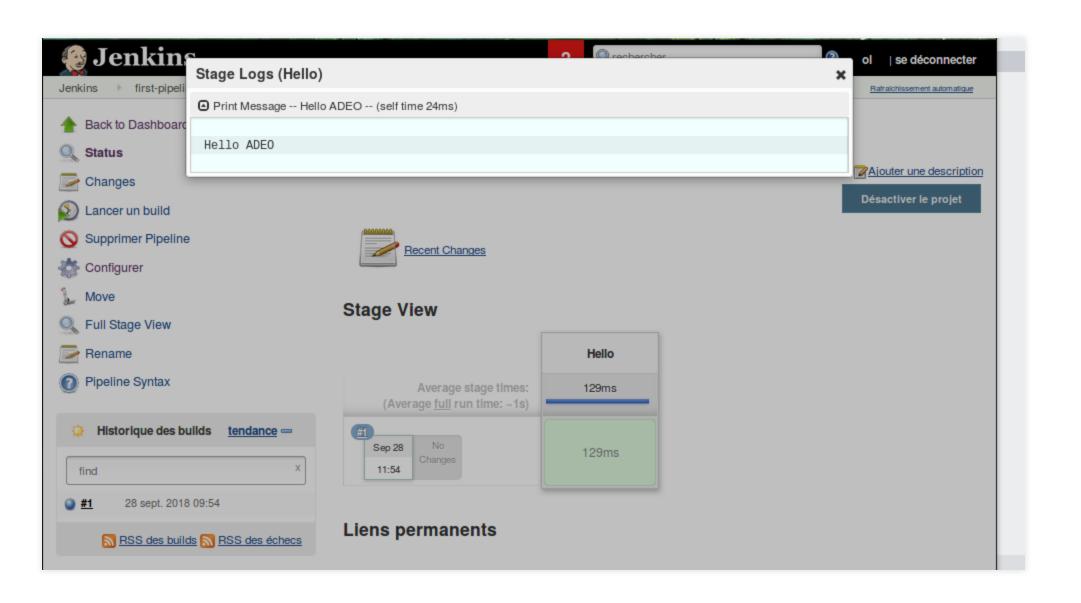


Correction









TP4



Objectifs

• Utiliser Git comme source pour un job pipeline



Sujet

• Créer un projet sur GitHub, et pousser un fichier tp4.groovy contenant :

```
node{
  stage('Hello'){
    echo "Hello ADEO"
  }
}
```

- Créer un job pipeline qui utilise le projet git (*Pipeline from SCM*) et renseigner le script tp4.groovy, ainsi que le secret Git créé au *TP 2*
- Exécuter le job manuellement



Correction

TP5



Objectifs

• Créer un scripted pipeline avec plusieurs stages



Sujet

- Créer un job tp5 de type pipeline en lien avec le projet git ;
- Créer un fichier tp5.groovy :
 - reprendre le stage Hello du tp4 ;
 - ajouter un nouveau stage pour afficher les variables d'environnement du node (fonction shell env);
- Pusher le code sur Github et lancer le job ;



Correction

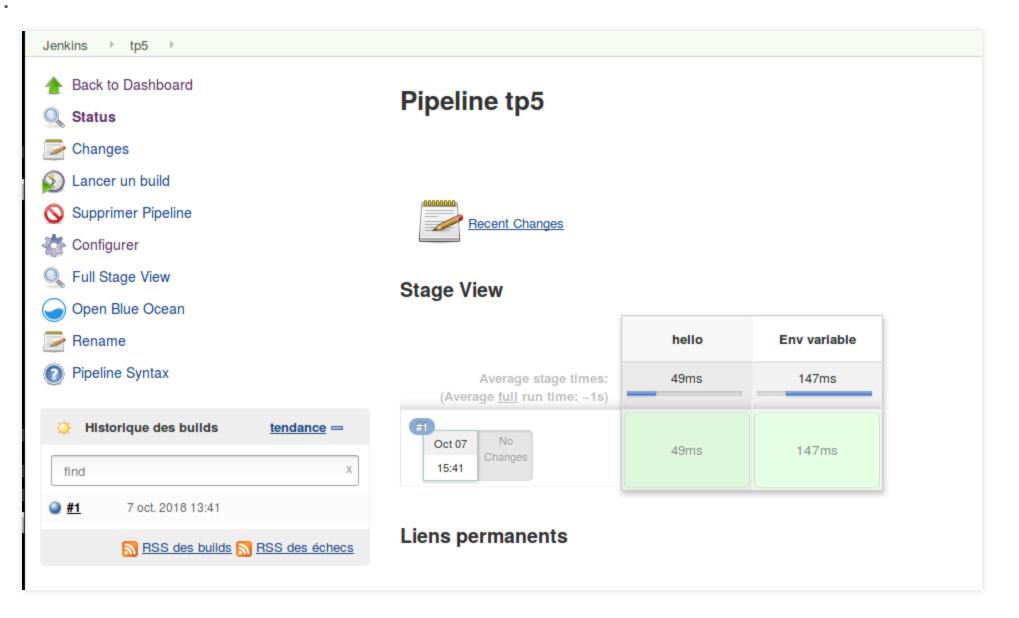
Fichier pipeline.groovy

```
node{
    stage('Hello'){
        echo "Hello ADEO"
    }

    stage ('Env variable'){
        sh 'echo Affiche toutes les variables environnement disponibles :'
        sh 'env'
    }
}
```



Résultat dans l'interface Jenkins :



TP6



Objectifs

- Apprendre à identifier des erreurs
- Utiliser des paramètres
- Utiliser des variables d'environnement
- Utiliser des secrets



Sujet

Télécharger <u>le tp6</u>:

- Créer un job pipeline pour le tp6
- Exécuter le job, et corriger les erreurs
- Modifier la section 1 Variables d'environnement pour afficher le numéro de build du job.
- Modifier la section *3 Récupération des secrets* pour afficher :
 - L'utilisateur et le mot de passe
 - Le chemin du fichier de secret



Correction

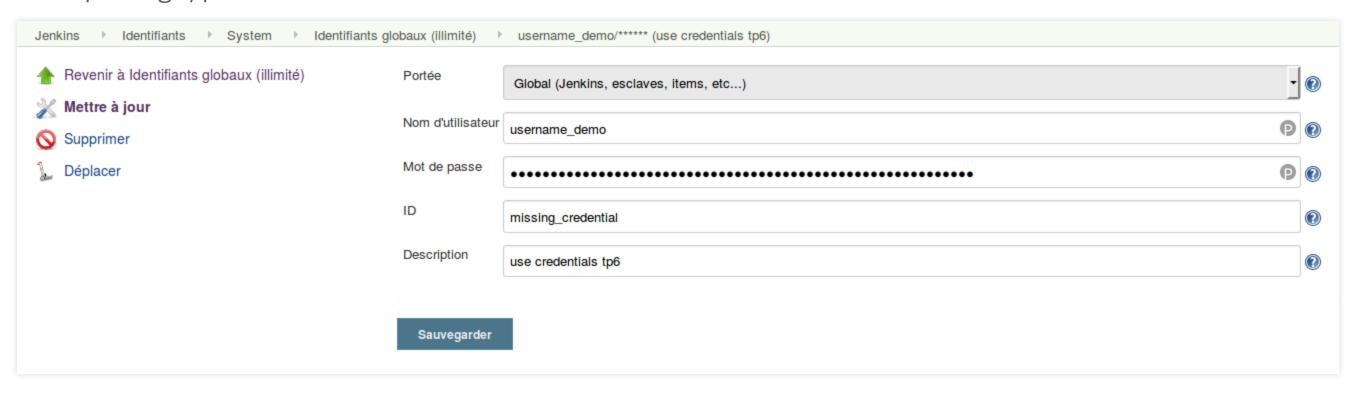
Création de variable globale:



Création d'un paramètre dans le job:

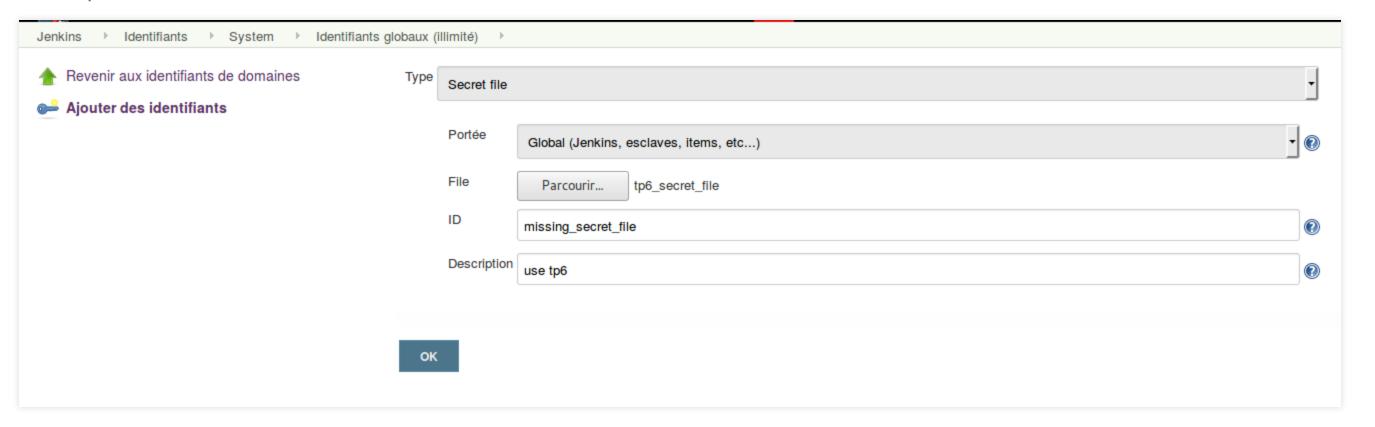


Création d'un crédential pour login/password:





Création d'un crédential pour un fichier secret:





Version corrigé du script tp6 :

<u>Télécharger</u> <u>tp6-stage1</u>

```
//
// Variable d'environnement
//
node {

stage('0- clean') {
    deleteDir()
        checkout scm
    }

stage ('1- Print Jenskins variables') {
        echo "Senv.SIORAL_GENKINS_VARIABLE"
    }

stage ('1- Print all env') {
        echo 'Affiche toutes les variables environnement disponibles :'
        sh 'env'
    }

stage ('1- BuildNumber') {
        echo 'Affichage du numero de build:' + BUILD_NUMBER
        sh 'echo SBUILD_NUMBER'
    }
}
```



<u>Télécharger</u> <u>tp6-stage2</u>



<u>Télécharger tp6-stage3</u>

```
// Credentials
node {
 stage('3- Récupération des crédentials'){
   withCredentials([
     usernamePassword(
         credentialsId: 'log_demo',
         usernameVariable: "DEMO_USERNAME",
         passwordVariable: "DEMO_PASS"
     file(
       credentialsId: 'secret_file_demo',
       variable: 'SECRET_FILE')
   ]){
     // Espace où sont accessibles les variables définies pour les crédentials
       echo -e "Affichage du username $DEMO_USERNAME \t doit être crypté"
       echo -e "Affichage du password $DEMO_PASS \t doit être crypté"
       echo -e "Affichage du path du fichier secret : $SECRET_FILE"
   } // end stage
 } // end withCredential, les variables ne sont plus accessible après
```

TP 7



Objectifs

- Gestion des erreurs
- Création d'une fonction



Sujet

<u>Télécharger</u> <u>le script tp7</u>

- 1. Créer un job pipeline pour le tp7;
- 2. Exécuter le job tp7 plusieurs fois, il génére une erreur si le numéro de build est impair ;
- 3. Modifier le stage « Gestion erreurs », remplacer les appels à la fonction groovy *println* par une fonction locale :
 - pour afficher le résultat du *if* ;
 - pour afficher un message d'erreur ;



Correction

Version final du script tp6 corrigé:

```
node {
 stage('Gestion erreur'){
   println "Commande avant le try/catch"
   try {
     println "Commande pouvant générer une erreur"
     def number = env.BUILD_NUMBER as Integer
     if (number%2) {
       //println "Number " + number + " is impair"
       print_result(number, "impair")
       // levee une exception
       throw new Exception()
     else {
       // println "Number "+ number +" is PAIR"
       print_result(number, "pair")
   } catch (Exception e) {
       // println "FAIL : la commande échoue, traiter ce cas"
       fail_message()
   } finally {
        println "Commande toujours exécutée"
 stage('End'){ println 'END'}
```

TP 8



Objectifs

• Utilisation de fonction partagée



Sujet

<u>Télécharger le script tp7</u>

- 1. Créer un job pipeline pour le tp8;
- 2. Dupliquer tp7.groovy en tp8.groovy. Au lieu de créer des fonctions locales pour afficher les messages,
 - créer une fonction partagée notify pour afficher les messages ;
 - utiliser le même projet git.



Correction

Utiliser le projet git comme source de code pour les fonctions partagées.

println "Check console output at \${env.BUILD_URL} \${env.JOB_NAME} [\${env.BUILD_NUMBER}]"

Ajouter un script groovy dans un dossier vars :

```
# à la racine du projet
mkdir vars

# placer le fichier notify.groovy

#!/usr/bin/env groovy

def call(status){
    println status + ": Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]':"
```



Configuration des fonctions partagées :



Version final du script tp8 corrigé :

<u>Télécharger le corrigé du tp8</u>