Conception d'applications internet

Servlets HTTP

Olivier Cailloux

LAMSADE, Université Paris-Dauphine

7 décembre 2015

LAMSADE
UMR CNRS 7243
laboratoire d'analyse et modélisation de systèmes pour l'aide à la décision

DAUPHINE
UNIVERSITÉ PARIS

PSL
RESEARCH
UNIVERSITY

Prérequis

Programmation Java théorique et appliquée ; ingénieurie logicielle théorique

- Exceptions
- Héritage
- Concept d'API
- *Programmation par contrat*
- XML
- *Maven*
- *Git*

Plus : annotations.

À faire

- Lisez (ou redirigez) vos e-mails @ Dauphine (pour les annonces)
- Installer les outils sur votre machine : Eclipse Mars Java EE, Java EE 7 (et Glassfish 4), Java 8 (OpenJDK)
- Choix d'un projet sur MyCourse
- Présentation (5 à 10 min) de vos idées
- Compte GitHub ou Bitbucket et commit initial : diapos

Présentations

À vous de jouer !

Logging

- `System.out.println` ?

Logging

- `System.out.println` ? Le conteneur redirige

Logging

- `System.out.println` ? Le conteneur redirige
- mieux : `LOGGER.warn("Missing parameter here!")`
- En Java EE comme en Java SE : cf. standard logging

Environnements et serveurs dans Eclipse

- Environnement Runtime : donne accès aux bibliothèques J. EE
- Eclipse : *Preferences* / *Server* / *Runtime Environments*
- Puis sélection par projet via *Targeted Runtime*

Serveurs

- Vue *Servers* dans Eclipse
- Liste les serveurs Java EE sur lesquels déployer les modules
- Utiliser : *New* ; *Add and Remove* ; *Publish*
- Sur un serveur : *Properties* puis *Switch location* pour le voir dans un projet spécial (donne plus d'options de configuration)

Git

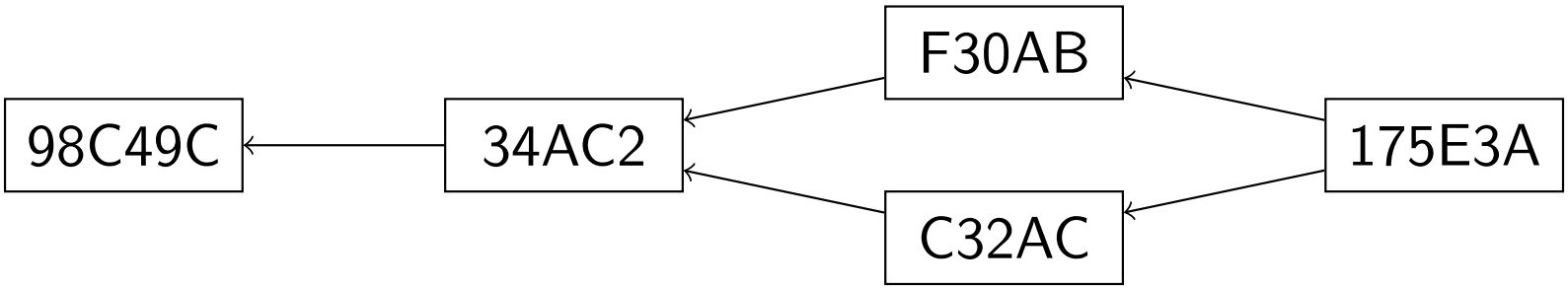
- Contrôle de version (VCS, SCM) : conserver l'historique
- Pour tous types de projet : code, images, présentations, article...
- VCS local, centralisé, distribué ?

Git

- Contrôle de version (VCS, SCM) : conserver l'historique
- Pour tous types de projet : code, images, présentations, article...
- VCS local, centralisé, distribué ?
- Centralisé : seulement sur un serveur distant
- Distribué : copie locale et distante

Commits et WD

- Histoire : un ensemble structuré (DAG) de « commits »
- Commit : identifié par un hash SHA-1
  - Contient : *blobs* (captures de fichiers) ; auteur…



- Histoire conservée *localement* dans `.git` à la racine du projet
- WD (« work dir ») : version du projet (fichiers et sous-répert.)

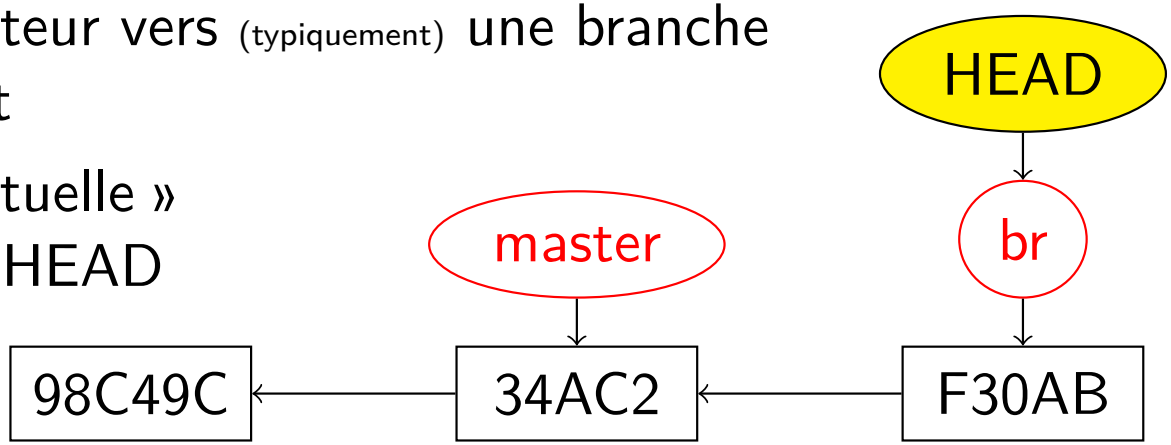
```
/root
    /.git
    /rép1
        /fich1
    /fich2
```

Préparer un commit

- WD ; index ; HEAD
- *Index* : changements à apporter au prochain commit
- fich. ∈ index (HEAD) : blob ds index (HEAD), peut être ≠ WD
- fichier (du WD) *untracked* : ∉ HEAD, ∉ index
- fichier *tracked* : blob dans HEAD ou dans index (ou les deux)
- `git add fichier` : blob mis dans index (« staged »)
- fichier *modified* : (∉ index ⇒ ≠ HEAD) ; (∈ index ⇒ ≠ index)
- *clean* WD : tous (sauf fichiers dans gitignore) tracked et unmodified
- `git status` : liste untracked, modified, staged ; WD clean ?
- `git status --short` (sauf merge conflict) : idx VS HEAD ; WD VS idx.
- `git diff` : WD VS index
- `git diff --staged` : index VS HEAD
- `git commit` : commenter et expédier ! (Renvoie son id SHA-1)
- `git commit -v` : voir l'index en détail

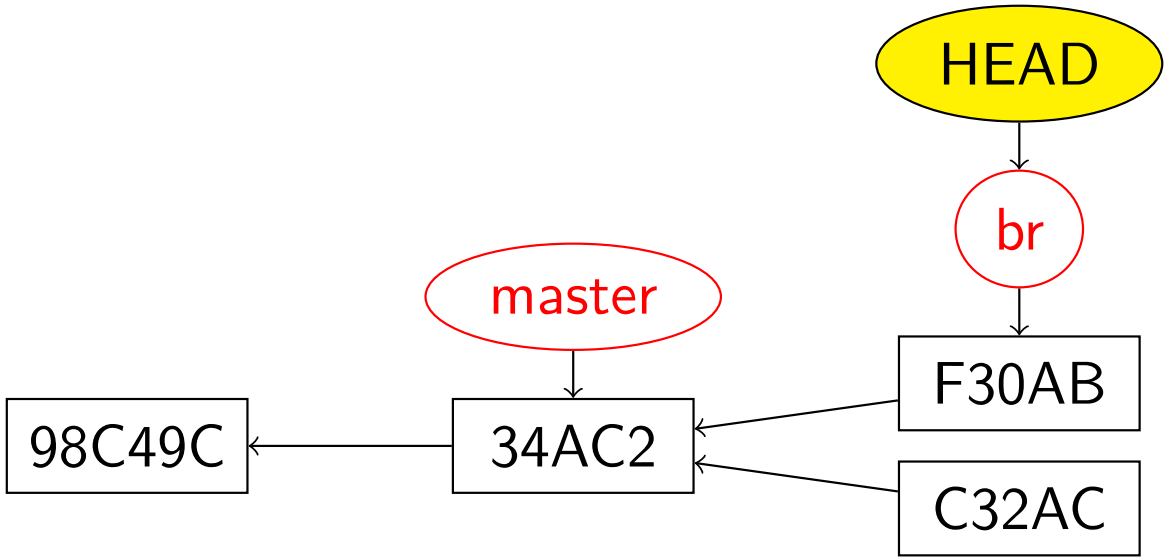
Branches et HEAD

- Branche : pointeur vers un commit
- HEAD : pointeur vers (typiquement) une branche et un commit
- Branche « actuelle » désignée par HEAD



- commit : avance HEAD et branche actuelle
- `git branch truc` : crée branche `truc`. HEAD inchangé !
- `git checkout truc` : change HEAD et éventuellement WD
- avoir un WD propre avant un checkout !
- `git log --graph --decorate --oneline --all`

Fusion de branches



- `git merge autrebranche` : fusionne changements de `autrebranche` dans branche actuelle
- Si `autrebranche` est en avant de l'actuelle : « fast-forward »
- Sinon, « merge conflict » possible. Modifier les fichiers à la main et les ajouter à l'index puis commit pour créer un merge.
- checkout d'un commit (ou tag) sans branche (detached head state) : lecture !

Serveurs distants

- Réf. distante (« remote ref ») : pointeurs vers branches et tags sur dépots distants
- « Remote-tracking branch » : branche locale correspondant à une branche distante et qui connait l'état de la branche distante correspondante la dernière fois qu'on l'a vue
- Remote « origin » supposé configuré ici
- `git branch -vv` : voir branches et correspondants distants
- `git fetch` : récupère les commits distants ; met à jour (ou crée) les références distantes
- `git push origin mabranche` : sinon, nouvelles branches restent locales
- `git remote show origin` : voir les réf. distantes
- Suivre une branche distante : checkout `origin/branche` ; créer branche locale ; `git branch --set-upstream-to origin/branch`

Divers

- Utilisez gitignore (modèles)
- Créez-vous une paire clé publique / privée
- Raccourcis : à éviter au début
- `git init` : dépôt vide dans rép. courant (rien n'est traqué)
- `git clone url` : cloner un dépôt (et non checkout !)
- `git stash` : WD ← HEAD
- `git tag -a montag` (tag annoté, recommandé) puis `git push origin montag`
- `git config --global` : écrit dans `~/.gitconfig`
- Indiquez propriété `user.name` (et `user.email`)
- Déterminer des révisions exemple : HEAD^1 pour parent de HEAD
- Alias
- Documentation
- GUI pour diff : `git difftool`
- GUI pour merge `git mergetool`

Exercices : servlets I

Ces exercices ont pour but de vous familiariser avec l'environnement de développement et vous permettre de vérifier que vous avez compris les concepts de base. Le but n'est atteint que si vous jouez le jeu : évitez de chercher comment faire sur internet ! Utilisez de l'aide seulement si vous êtes bloqués.

- « Installer » GlassFish (copier `/usr/local/glassfish-4.1`)
- Démarrer votre serveur (cf. `bin/`, `http://localhost:8080`, `http://localhost:4848`)
- Désactiver l'écoute extérieure
- Lire les logs du serveur
- Ajouter dans Eclipse un environnement runtime Java EE rudimentaire, « J2EE Preview »

Exercices : servlets II

- Créer un projet web dynamique sans runtime puis le modifier pour qu'il vise ce runtime. Vérifier que Eclipse ajoute les bibliothèques en dépendances (lesquelles ?)
- Créer un servlet
- Vérifier que vous avez accès à la javadoc (par exemple sur `javax.servlet.annotation.WebServlet` et `javax.servlet.http.HttpServlet`).
- Programmer réponse : "ça fait 0"
- Exporter le module dans une archive déployable (extension ?)
- Vérifier si les bibliothèques ajoutées automatiquement par eclipse ont été incluses dans l'archive exportée. Expliquer pourquoi (pas).
- Déployer le module sur le serveur à la main
- Envoyer une requête GET et observer "ça fait 0"

Exercices : servlets III

- Se féliciter [1]
- Ajouter un runtime à eclipse : J2EE Runtime library, référencer les bibliothèques installées dans glassfish. Modifier votre projet existant pour utiliser ce runtime (voir propriétés du projet). Qu'est-ce que ça change ?
- Créer un nouveau serveur de type J2EE Preview dans Eclipse. Expliquer pourquoi il n'y a pas « J2EE Runtime library » dans la liste de serveurs qui peuvent être créés de cette manière. Déployer votre servlet depuis eclipse vers ce nouveau serveur. Vérifier que votre servlet fonctionne depuis votre navigateur, ansi que depuis eclipse (*Run* / *Run*). [2]
- Installer Glassfish Tools depuis Eclipse Marketplace
- Utiliser glassfish comme runtime. Voir ce que ça change. Déployer vers glassfish et vérifier que le servlet fonctionne toujours

Exercices : servlets IV

- Envoyer une information de log indiquant le résultat du calcul avant son renvoi à l'expéditeur
- Renvoyer une réponse HTML bien formée : <html><body> Ça fait... </body></html>. Comment s'assurer que le client web va bien interpréter de l'HTML ?
- Renvoyer ce même contenu mais en texte pur, de façon à ce que ces balises s'affichent sur le client web. Comment s'assurer de l'affichage correct des caractères non ascii, Ç par exemple ?

*[3] Est-il possible de déployer votre module web comme une archive `.ear` ? Si oui, essayer de le faire, si non, expliquer pourquoi.

- Accepter deux paramètres dans le servlet : `add1` et `add2`. Renvoyer "ça fait " et l'addition des paramètres
- Renvoyer une erreur s'il manque un paramètre (indice : voir la javadoc de `HttpServletResponse`).

Exercices : servlets V

* Comment être sûr du type d'erreur à renvoyer ?

- Ajouter au module une page HTML statique `index.html` qui dit « Hello ! » lorsqu'on la visite. Où sera-t-elle placée dans l'archive ?
- (Commencer à) développer un servlet qui a sa place dans votre projet

---

1. Cette étape importante est à répéter à l'issue de chaque exercice qui vous a posé une difficulté.

2. Il semble qu'un bug dans J2EE Preview empêche parfois l'accès à votre servlet via ce serveur. Vérifiez simplement dans ce cas que vous pouvez accéder à `http://server/context-root/`.

3. Les astérisques indiquent des exercices plus difficiles, à essayer de faire pour gagner des points de prestige.

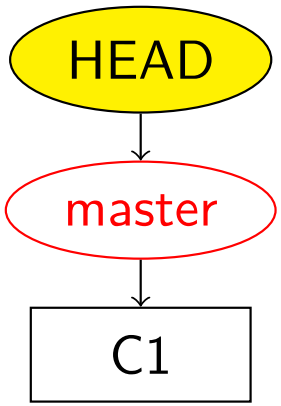
Exercices : Git I

Git en local

- Définir globalement (au moins) `user.name`. Vérifier avec `git config --list`.
- Créer un répertoire projet et dedans un fichier `début.txt` contenant `"coucou"`.
- Initialiser un dépôt git dans ce projet.
- Placer `début.txt` dans l'index. Modifier `début.txt` pour qu'il contienne `"coucou2"`. Visualiser la différence sur ce fichier entre la version WD, index, et dépôt. Faire en sorte que le blob dans l'index contienne bien `"coucou2"`.

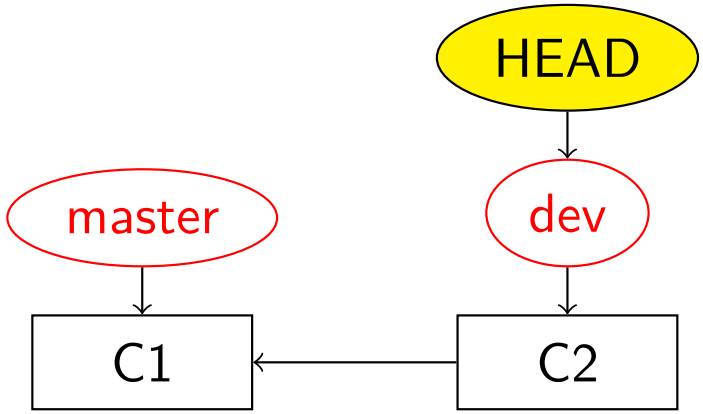
Exercices : Git II

- Effectuer un premier commit, qui contiendra uniquement `début.txt`. À l'issue de ce commit, vérifier que vous obtenez l'historique suivant.



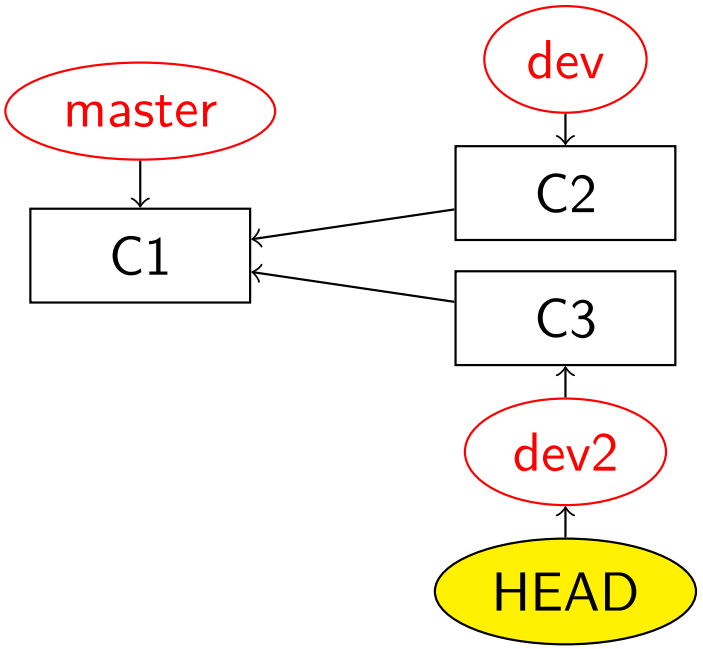
Exercices : Git III

- Vous avez maintenant une idée audacieuse pour résoudre un problème dans votre projet. Comme vous n'êtes pas sûr de sa pertinence, vous désirez placer vos changements dans une nouvelle branche en attendant d'y réfléchir. Créer une branche `"dev"` ; y commettre un fichier `audacieux.txt` (en plus de `début.txt`, inchangé) contenant `"approche 1"`. Votre historique doit maintenant être celui-ci (vérifier !).



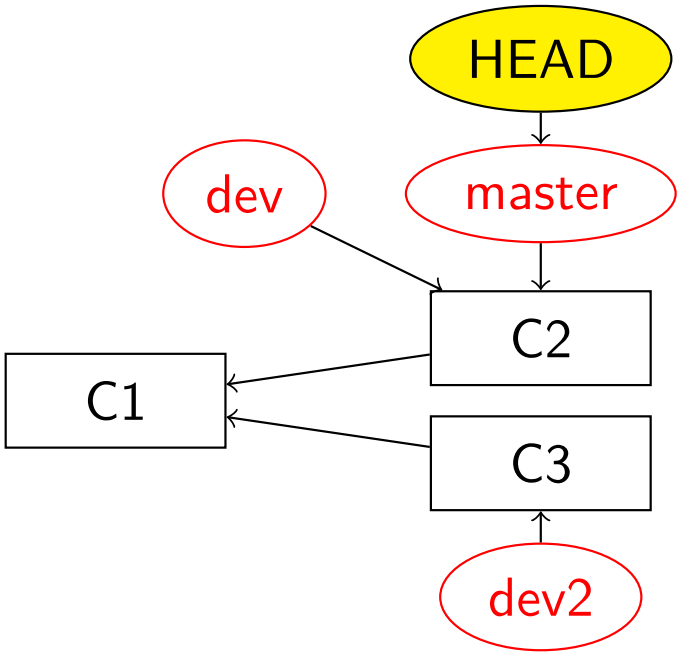
Exercices : Git IV

- À l'issue de ce travail harrassant, il vous vient une idée alternative. N'étant toujours pas sûr de la valeur de votre première idée (dans `dev`), vous repartirez de `master` pour l'implémenter. Depuis `master`, créer une branche `dev2`, et y commettre (en plus de `début.txt`, inchangé) un fichier `audacieux.txt` contenant `"approche alternative"`. Vérifier ensuite votre historique.



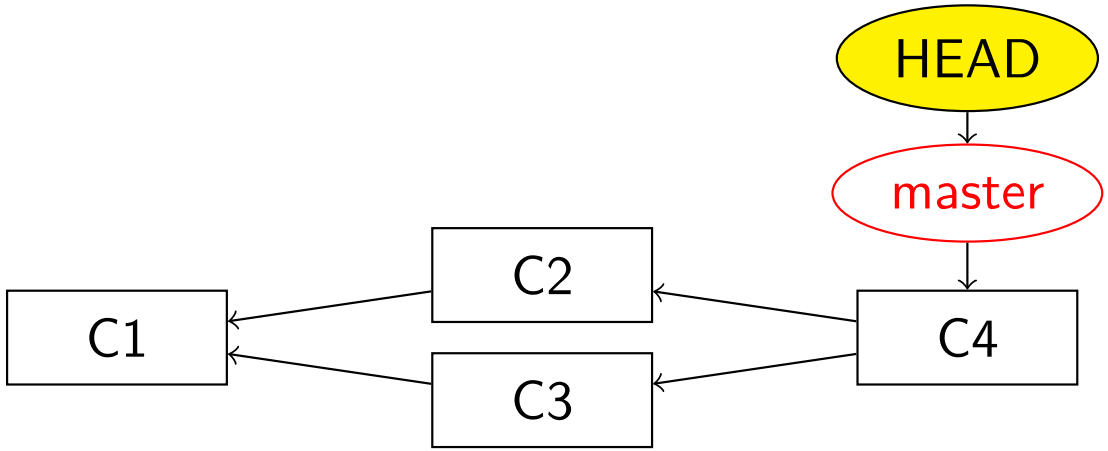
Exercices : Git V

- À la réflexion, votre première idée est bonne. L'intégrer dans `master` pour obtenir l'historique suivant. Prédire si vous obtiendrez un fast-forward et vérifier.



Exercices : Git VI

- Tout bien réfléchi vous aimez également votre deuxième idée. L'intégrer à son tour dans `master` et obtenir cet historique. Quel problème allez-vous rencontrer, ce faisant ?



- Imaginons qu'on aurait d'abord intégré `dev2` à `master` (ceci aurait-il produit un fast-forward ?) puis `dev` au résultat. Quel aurait été le résultat final ?

Git distant

- Cloner votre dépôt central pour le projet de ce cours.

Exercices : Git VII

- Le clonage vous a créé un pointeur vers un serveur distant `origin`, et une « remote-tracking branch » `master`. Voir où pointent `origin`, `master` et `origin/master`.
- Ajouter un fichier `"macontrib.txt"` contenant votre prénom à votre index local. Commettre dans votre dépôt git local. L'envoyer au dépôt distant. Vérifier (via l'interface web) qu'il s'y trouve et que le commit est associé à votre nom.
- Quand un collègue a fait de même, vous pouvez rapatrier sa modification en local. Après l'avoir fait, prédire où vont pointer `master` et `origin/master` et vérifier.
- Si un collègue a publié sa modification avant la vôtre, quel va être votre problème ? Comment le résoudre ? Si vous êtes le premier à publier la modification, allez aider vos collègues à publier la leur !

À faire

Avant le 14 décembre

- Terminer les exercices Servlets et Git

Avant le 21 janvier

+[a] Chaque binôme : *au moins* un servlet utile à votre projet

\+ Au moins un push par personne dans le projet

a. Le + indique que cet aspect intervient dans la note

Chaque commit doit indiquer clairement le ou les auteurs

- Utilisez user.name
- Indiquez dans le commentaire du commit le binôme éventuel

Licence