WAS Liberty+DevOpsツールで実現する
インフラ構築自動化ガイド

DevOpsツール説明

目次

1. git
2. Maven
3. Jenkins
4. Arquillian

git

Tool：git

- ツール概要
  - 何ができるのか？
  - どこで使うか？
  - Libertyとの連携は？
  - 仕組み / アーキテクチャ
- 使用手順
  - インストール
  - セットアップ

ツール概要

1. 何ができるのか？
2. どこで使うか？
3. Libertyとの連携は？
4. 仕組み / アーキテクチャ

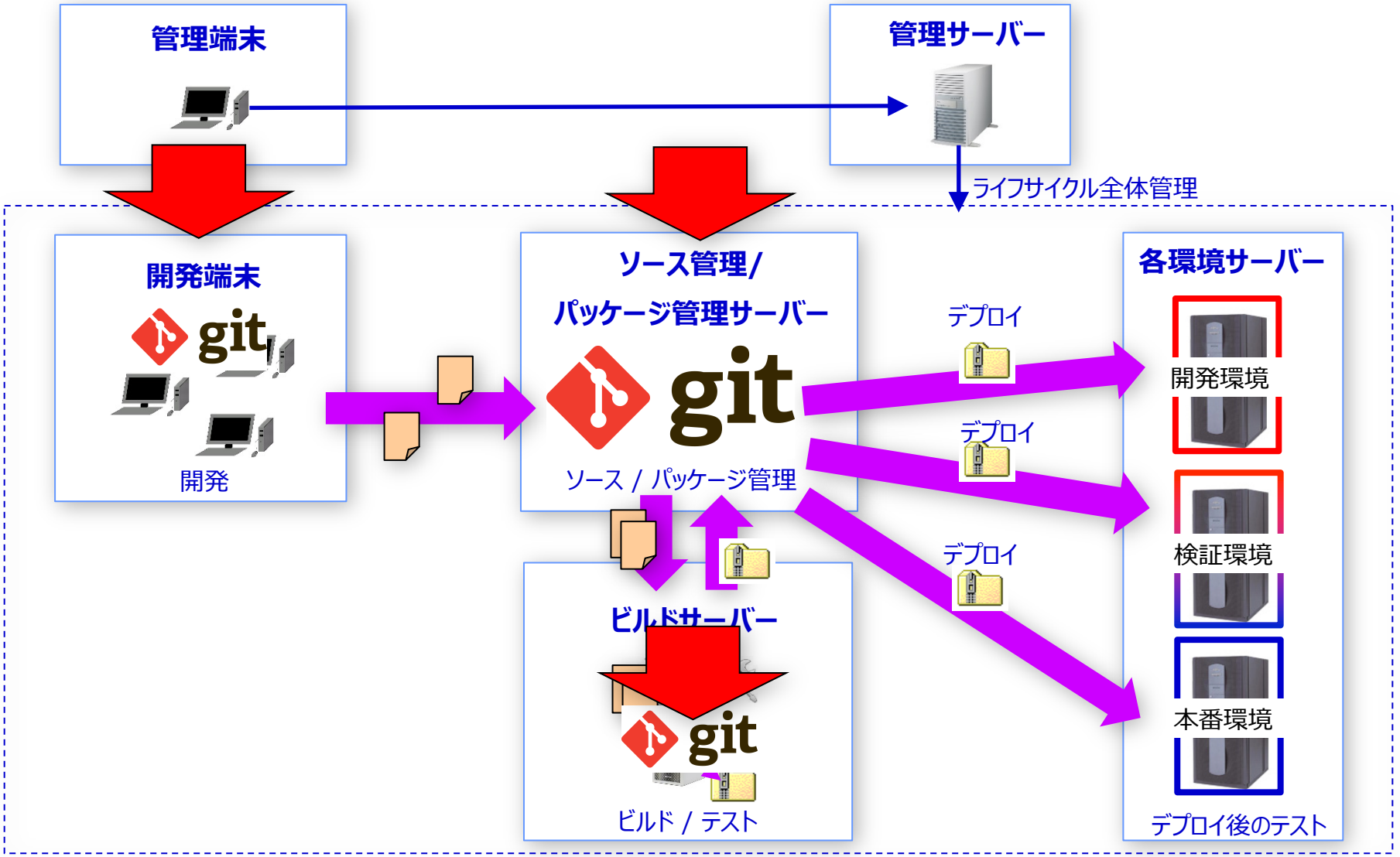
1. Git – 何ができるのか？

- 分散型バージョン管理システム
  - Linuxカーネルのソースコード管理に用いるためにリーナス・トーバルズによって開発され、それ以降ほかの多くのプロジェクトで採用
  - gitでは、各ユーザのワーキングディレクトリに、全履歴を含んだリポジトリの完全な複製が作られる。したがって、ネットワークにアクセスできないなどの理由で中心リポジトリにアクセスできない環境でも、履歴の調査や変更の記録といったほとんどの作業を行うことができる。これが「分散型」と呼ばれる理由である。

（wikipedia より）

- こちらもどうぞ！
  - Git のコマンドだけでなく、その仕組みを学ぶ
    - https://www.ibm.com/developerworks/jp/devops/library/d-learn-workings-git/

2. git – どこで使うか？

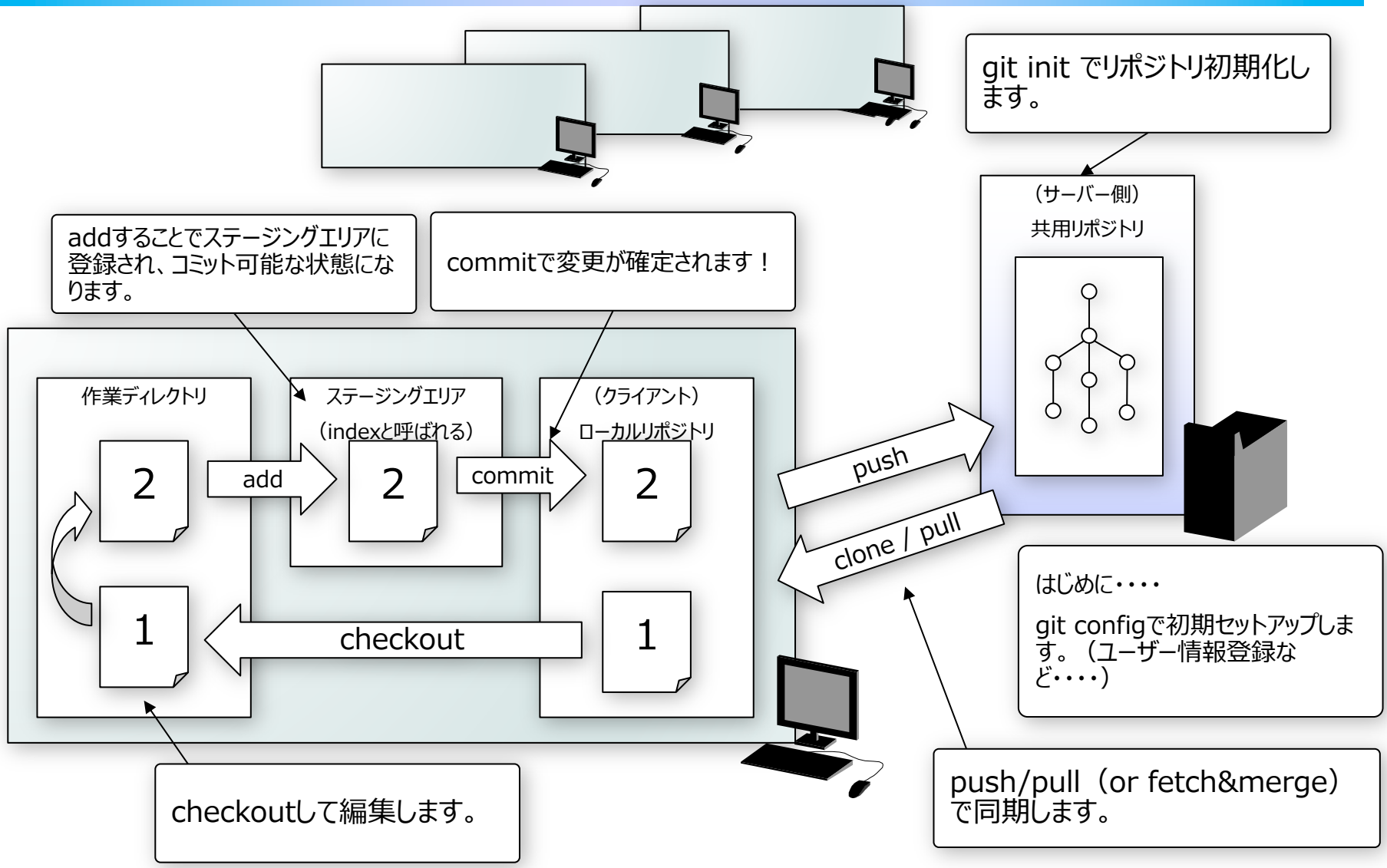


3. git – Liberty との連携は？

- git と Liberty を直接連携するgitのプラグインのようなものは無さそう。
- git で Liberty の設定ファイル群を管理する、という関係になります。

  - また・・・Liberty関連のコードやサンプルなどの殆どはGitHubに置かれています。
  - https://github.com/wasdev

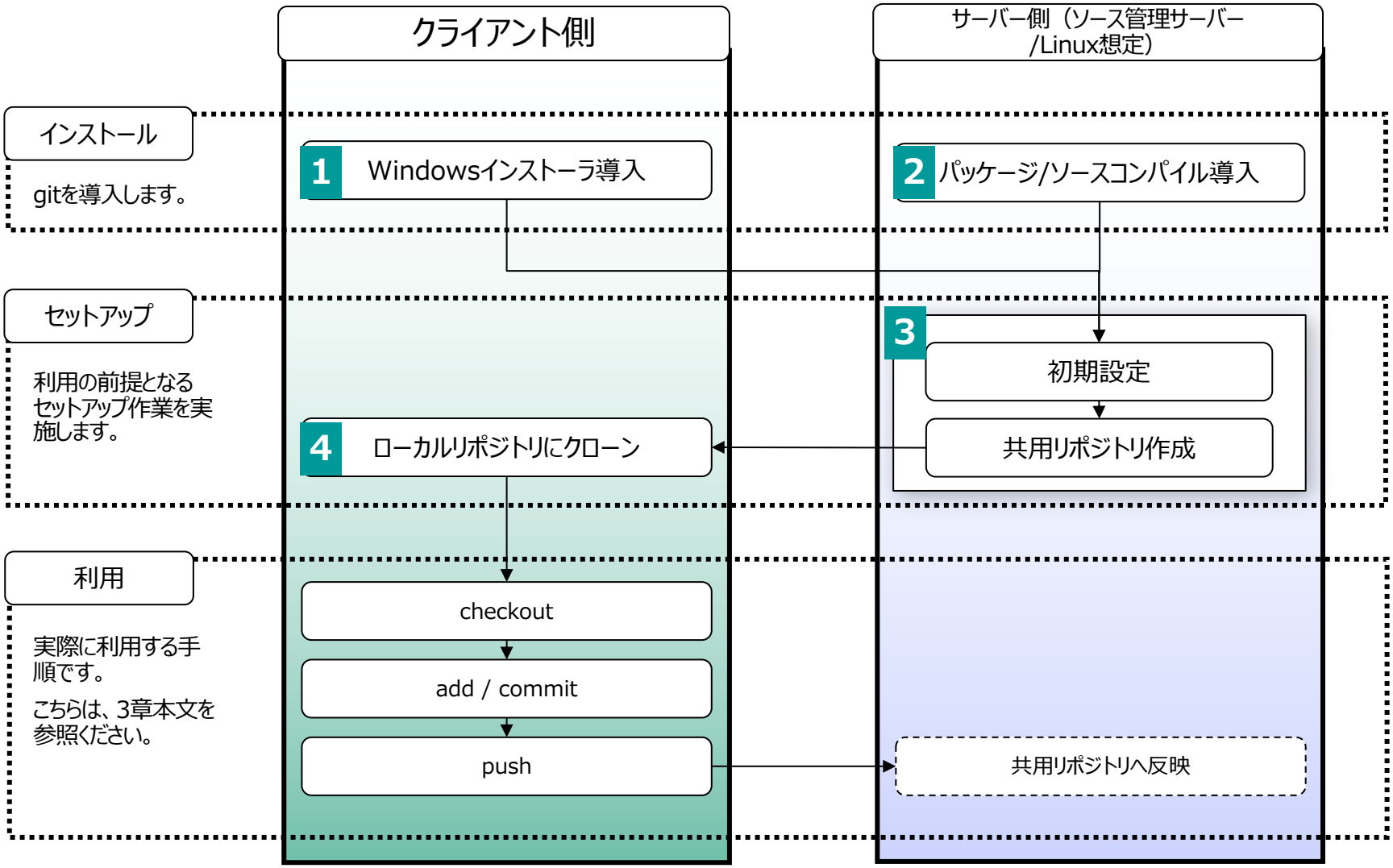
4. git - 仕組み/アーキテクチャー



使用手順

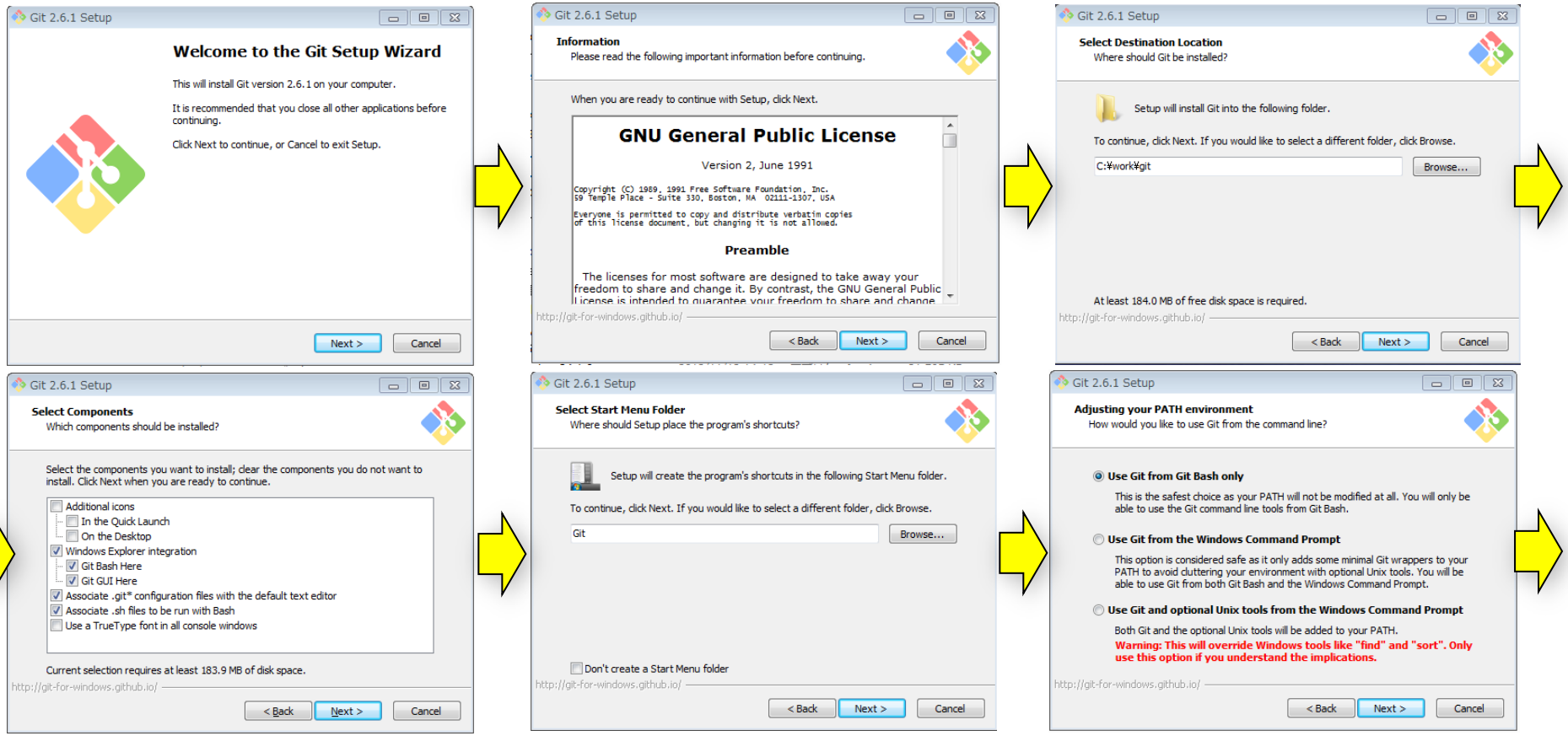
1. 全体の流れ
2. インストール
3. 初期セットアップ

1. 全体の流れ



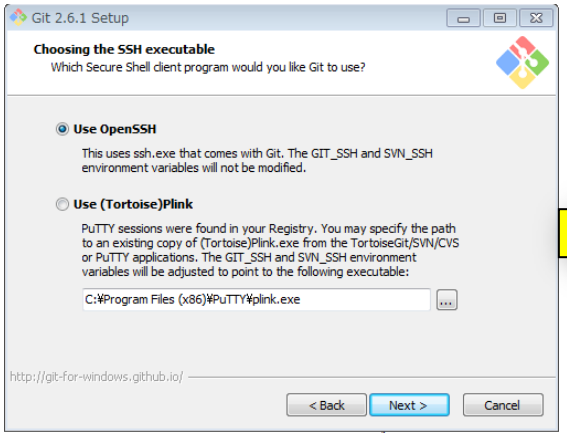
1 インストール！（クライアント側 1/2）

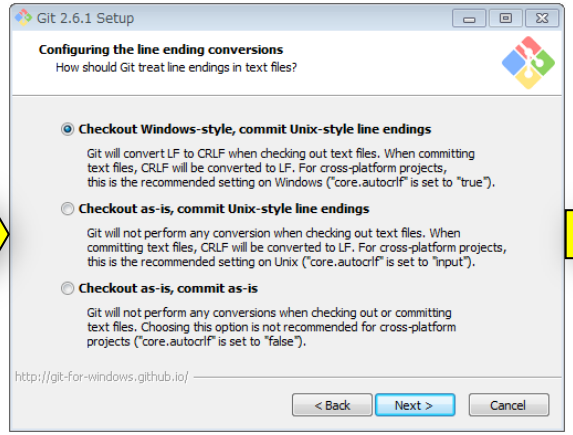
- ダウンロードします。
  - http://git-scm.com/downloads
- インストーラを起動して、wizardに従い導入します。

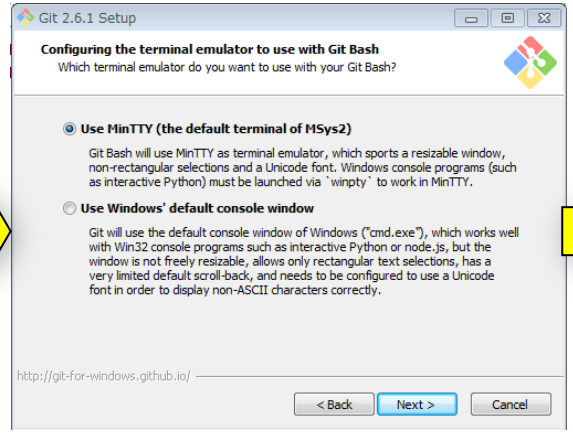


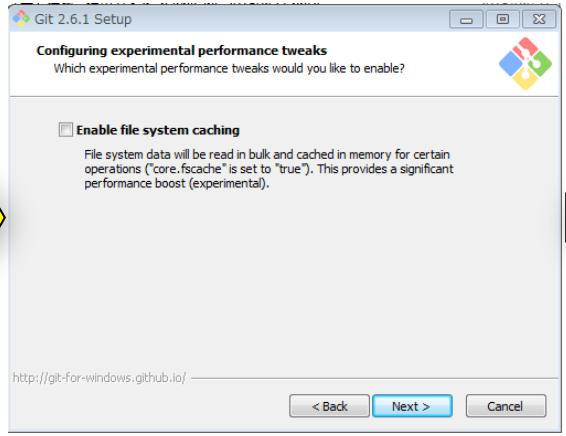
1 インストール！（クライアント側 2/2）

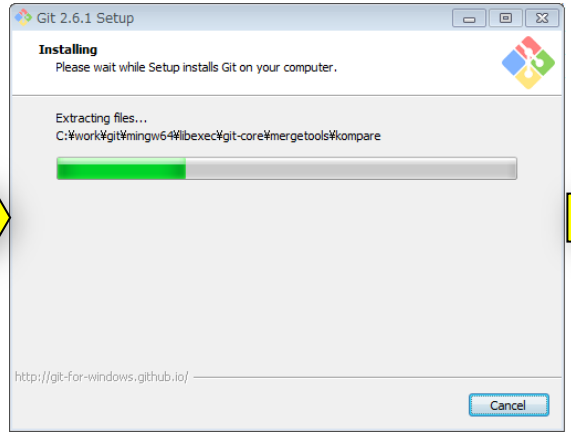
- wizardに従い導入します。

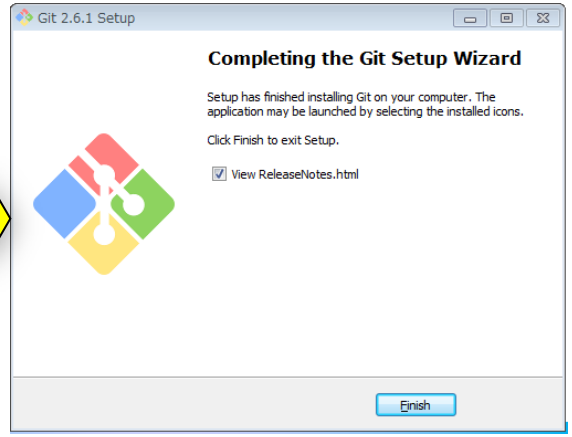












2 インストール！（サーバー側）

- 前提パッケージを導入します。
  - gitのダウンロードサイトでは、各ディストリビューションのパッケージマネージャでの導入が記載されていますが、ここではソースからコンパイルして導入する方法を紹介します。

```
[wasadmin@scm ~]$ yum -y install curl-devel expat-devel gettext-devel openssl-devel zlib-devel perl-ExtUtils-MakeMaker
```

- ソースコードをダウンロードして解凍します。

```
[wasadmin@scm ~]$ wget https://www.kernel.org/pub/software/scm/git/git-2.6.1.tar.gz
[wasadmin@scm ~]$ tar zxvf git-2.6.1.tar.gz
```

- インストールします。

```
[wasadmin@scm ~]$ ./configure --prefix=/opt/git
[wasadmin@scm ~]$ make
[wasadmin@scm ~]$ make install
```

- 確認してみます。

```
[wasadmin@scm ~]$ [wasadmin@scm git]$ git --version
git version 2.6.1
```

3 初期セットアップ（サーバー側 1/4）

- ユーザー等の設定
  - 操作を行うユーザーの設定を行います。あわせて、ui文字色の自動設定も行っています。

```
[wasadmin@scm ~]$ git config --global user.name wasadmin
[wasadmin@scm ~]$ git config --global user.email eb82649@jp.ibm.com
[wasadmin@scm ~]$ git config --global color.ui.auto
[wasadmin@scm ~]$ git config -l
user.name=wasadmin
user.email=eb82649@jp.ibm.com
```

- Liberty導入ディレクトリをリポジトリとして初期化
  - すでにSCMサーバー上に導入し、templateServerというサーバーを作成済みのLiberty環境（/opt/IBM/Liberty/build/）にて、以下のコマンドを実行し、Libertyディレクトリをリポジトリ化します。
  - Gitでは、共有用のリポジトリはbareリポジトリとして作成する必要があります。Bareリポジトリは、workディレクトリを持ちませんので、addやcommitができません。したがって、いったん/opt/IBM/Liberty/buildディレクトリにLibertyを導入し、ここからcloneして/opt/IBM/Liberty/liberty-shared ディレクトリに共有用のリポジトリを作成する、という手順で共有リポジトリを作成します。

```
[wasadmin@scm build]$ pwd
/opt/IBM/Liberty/build
[wasadmin@scm build]$ git init
Initialized empty Git repository in /opt/IBM/Liberty/build/.git/
```

3 初期セットアップ（サーバー側 2/4）

- リポジトリ初期化後、状態を確認してみます。

```
[wasadmin@scm build]$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        wlp/

nothing added to commit but untracked files present (use "git add" to track)
```

この状態では、addを行っていないので、Untrackedという状態で認識されています。

- wlp以下を丸ごとaddします。
  - wlp以下をaddして、現在の状態を登録します。

```
[wasadmin@scm build]$ git add .
[wasadmin@scm build]$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   wlp/CHANGES.TXT
        new file:   wlp/Copyright.txt
        new file:   wlp/README.TXT
                :
        new file:   wlp/usr/servers/.classCache/C280M4F0A64P_liberty-wasadmin_G33
        new file:   wlp/usr/servers/.logs/status.log
        new file:   wlp/usr/servers/templateServer/server.env
        new file:   wlp/usr/servers/templateServer/server.xml
        new file:   wlp/usr/servers/templateServer/workarea/.sLock
```

addすることによって、commit可能な状態としてwlp配下が認識されるようになりました。

3 初期セットアップ（サーバー側 3/4）

- wlp以下を丸ごとcommitします。
  - 続いて、"initial commit"というコメント付でcommitします。

```
[wasadmin@scm build]$ git commit -m "initial commit"
[master (root-commit) 499f997] initial commit
 1740 files changed, 138275 insertions(+)
 create mode 100644 wlp/CHANGES.TXT
 create mode 100644 wlp/Copyright.txt
 create mode 100644 wlp/README.TXT
                 :
 create mode 100644 wlp/usr/servers/.classCache/C280M4F0A64P_liberty-wasadmin_G33
 create mode 100644 wlp/usr/servers/.logs/status.log
 create mode 100644 wlp/usr/servers/templateServer/server.env
 create mode 100644 wlp/usr/servers/templateServer/server.xml
 create mode 100644 wlp/usr/servers/templateServer/workarea/.sLock
```

実際にcommitが行われています。

commitされたファイルのリストされます。

- 確認してみます。
  - Commit後の状態を確認します。

```
[wasadmin@scm build]$ git status
On branch master
nothing to commit, working directory clean

[wasadmin@scm scm]$ git log
commit 499f997cd754a271a08cfe8dc4b42ef0f507aa25
Author: wasadmin <eb82649@jp.ibm.com>
Date:   Tue Oct 20 20:22:14 2015 +0900

    initial commit
```

再度ステータスを確認すると、変更されたファイルはすべてcommitされた状態のため、リストの出力はありません。

ログを確認すると、先ほどのcommit操作が記録されているのがわかります。

4 初期セットアップ（サーバー側 4/4）

- これを、共有用のリポジトリとしてセットアップします。

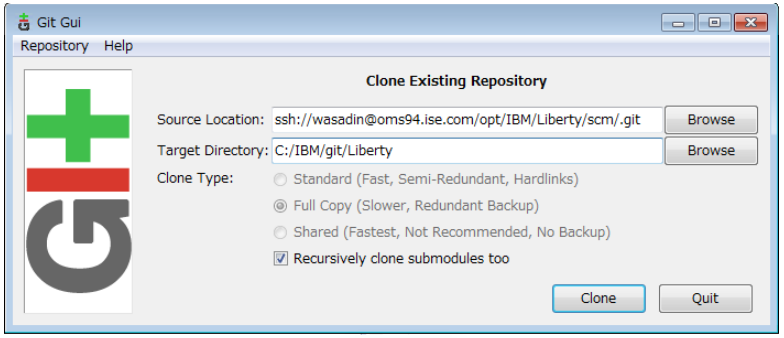
```
[wasadmin@scm build]$ git clone --bare /opt/IBM/Liberty/build/ /opt/IBM/Liberty/liberty-shared/
Cloning into bare repository '/opt/IBM/Liberty/liberty-shared'...
done.
```

- 確認してみます。
  - Commit後の状態を確認します。

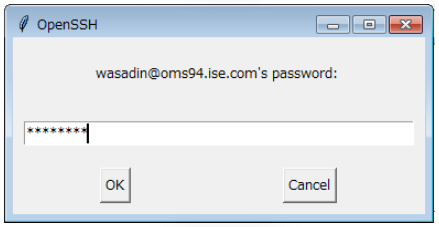
```
[wasadmin@scm build]$ git status
On branch master
nothing to commit, working directory clean

[wasadmin@scm scm]$ git log
commit 499f997cd754a271a08cfe8dc4b42ef0f507aa25
Author: wasadmin <eb82649@jp.ibm.com>
Date:   Tue Oct 20 20:22:14 2015 +0900

    initial commit
```

4 ローカルリポジトリにクローン（クライアント側 git gui編）

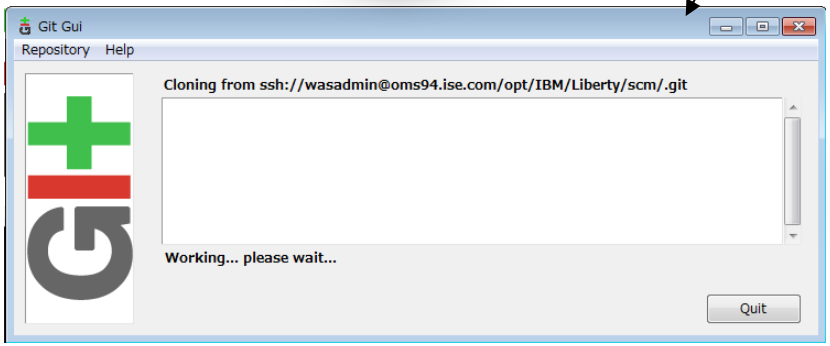
- クライアント側での運用方法にあわせて、git gui（git導入でついてくるツール）もしくは eclipse いずれかを選択してクローン操作を実行します。



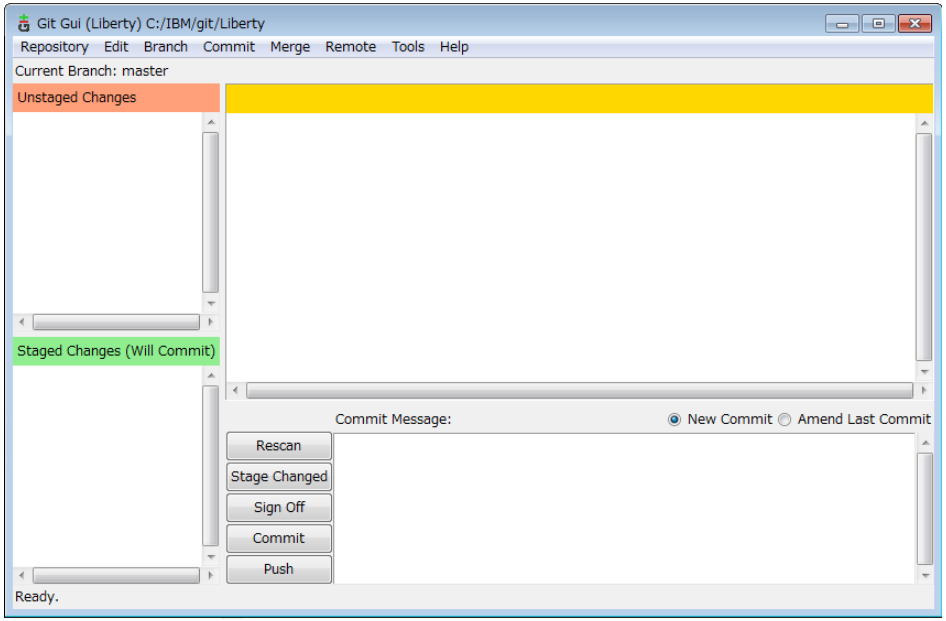
接続先の共用リポジトリのURLと、クローン先のローカルのディレクトリパスを指定します。



SSH接続のため、パスワードを入力します。進捗状況が表示されます。

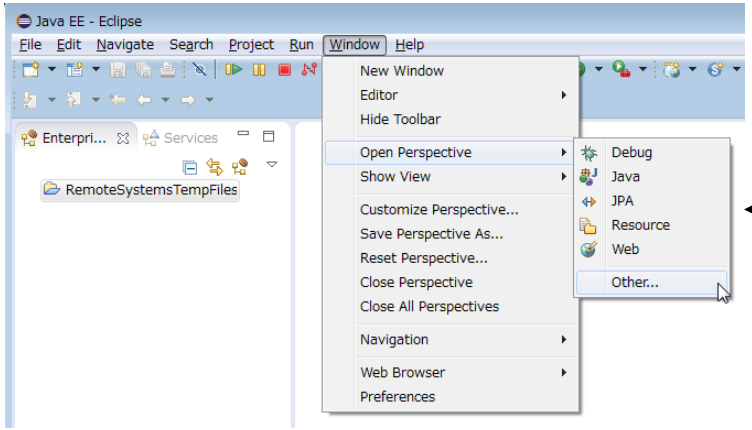


以下の通り、サーバー上のリポジトリがローカルにクローンされます。



4 ローカルリポジトリにクローン（eclipse プラグイン編）

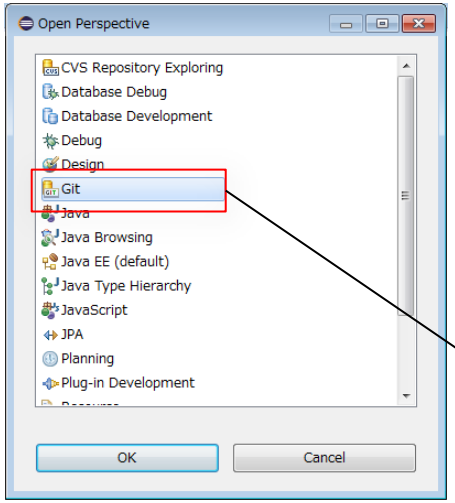
- Eclipseでの操作を考えると・・・
  - Libertyプロファイルでのサーバー設定を行う場合は、クライアント側でEclipseを介して実施するケースが多いかと思われます。
  - 以下は、Eclipseでegitを使用してリポジトリのセットアップを行う手順です。

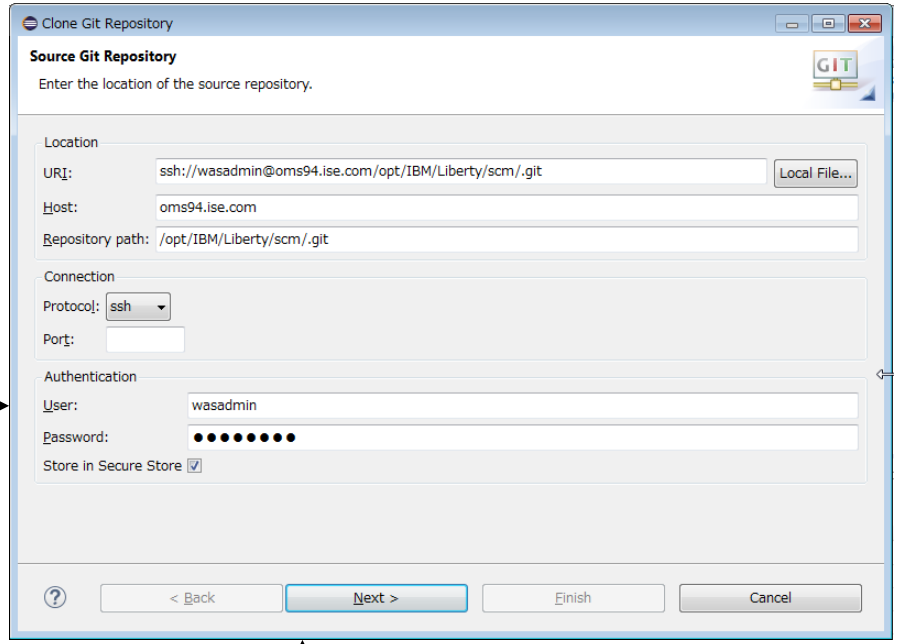


メニューから、gitのパースペクティブを開きます。

4 ローカルリポジトリにクローン（eclipse プラグイン編）

- Eclipseでの操作は・・・egit（eclipseのgitプラグイン）で！

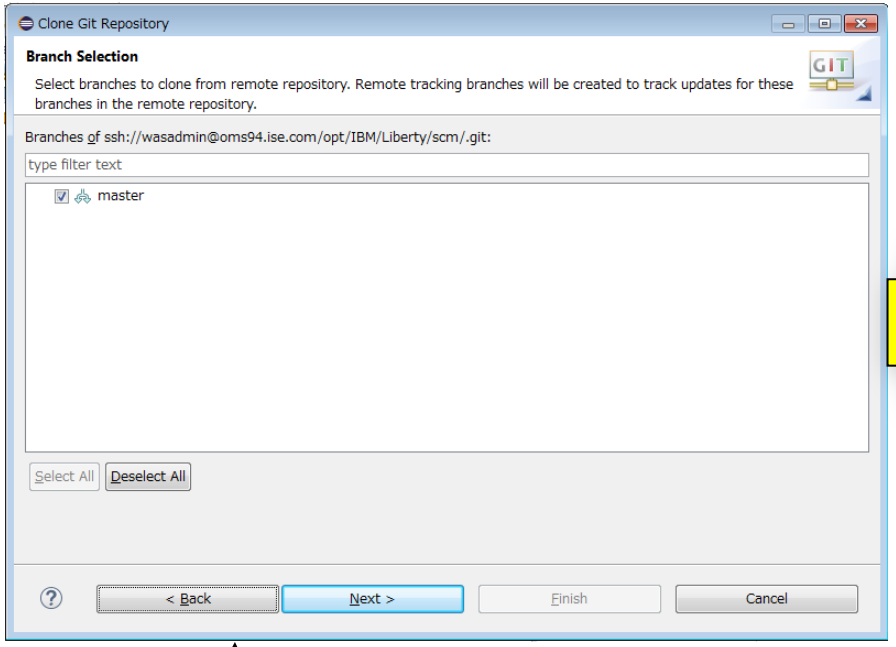


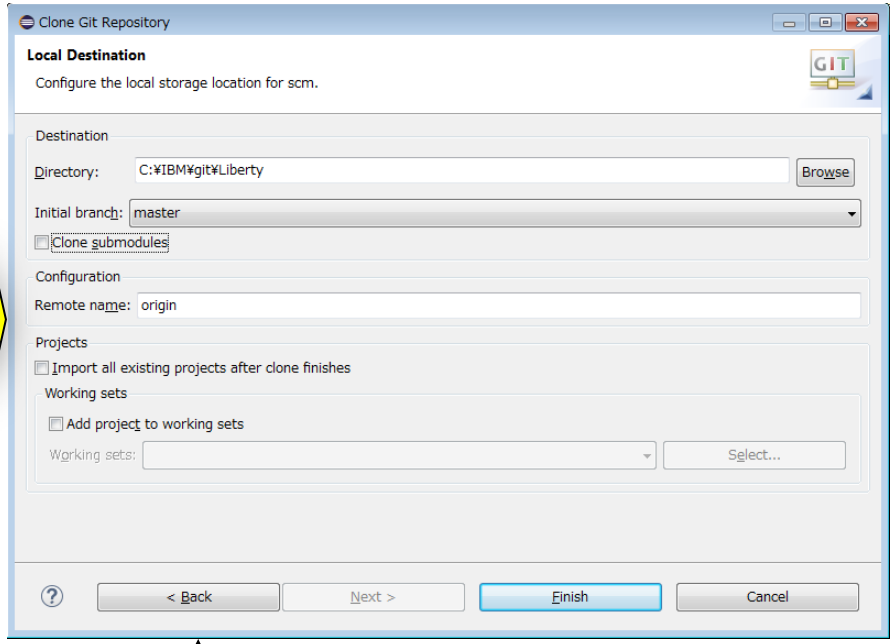


Gitリポジトリのクローンを選択します。

接続先の共用リポジトリのURLを指定します。

4 ローカルリポジトリにクローン（eclipse プラグイン編）

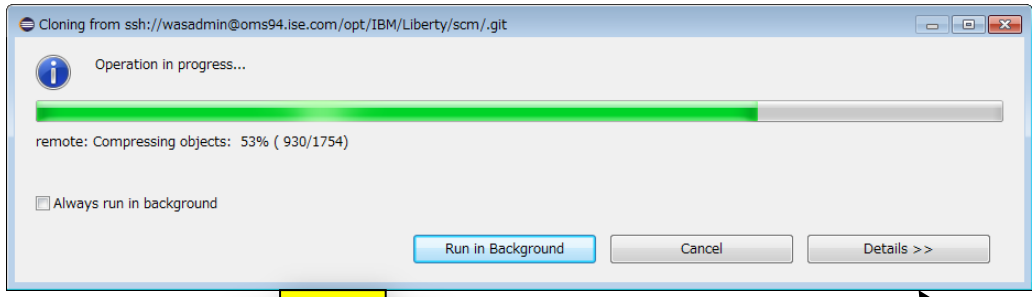




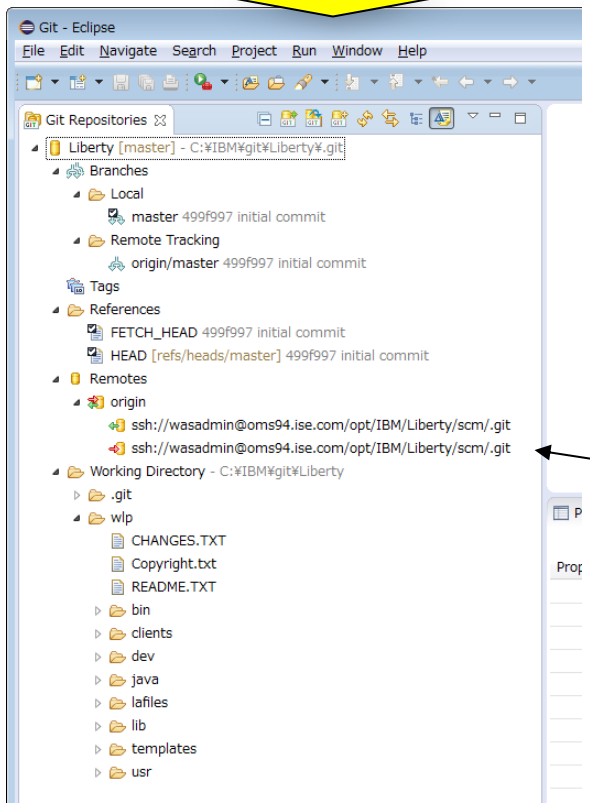
ブランチを選択します。
まだ作成していませんので、デフォルトのmasterのみ存在している状態です。

クローン先のローカルのディレクトリを指定します。

4 ローカルリポジトリにクローン（eclipse プラグイン編）

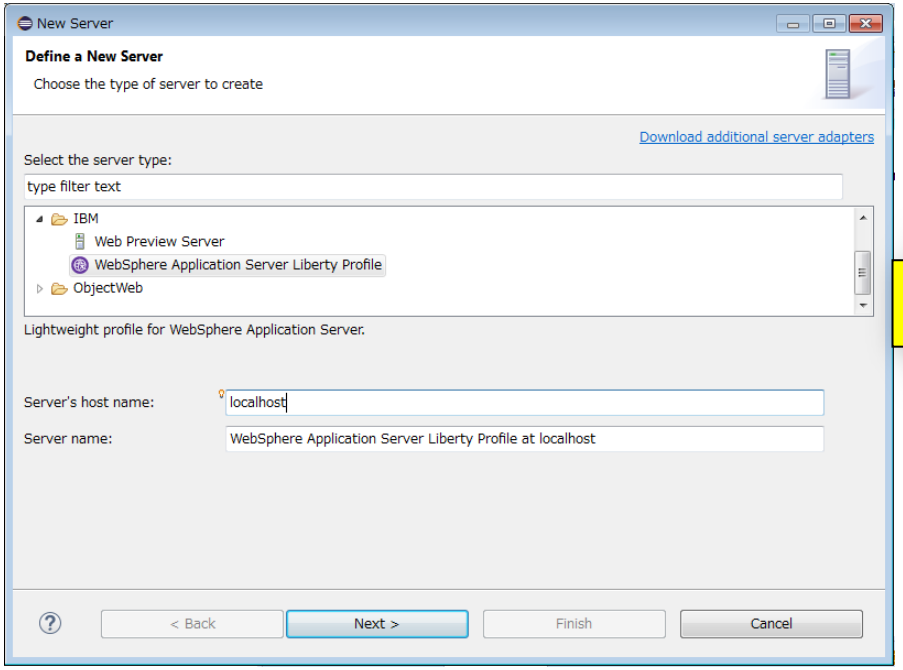


進捗状況が表示されて・・・・・

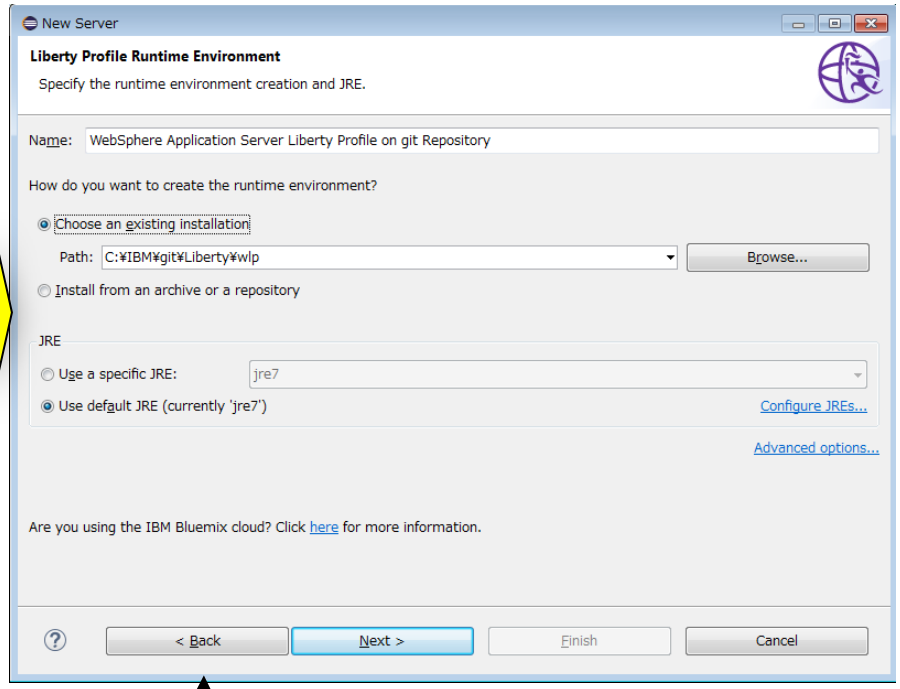


無事ローカルにクローンが作成されました。

4 ローカルリポジトリにクローン（eclipse プラグイン編）

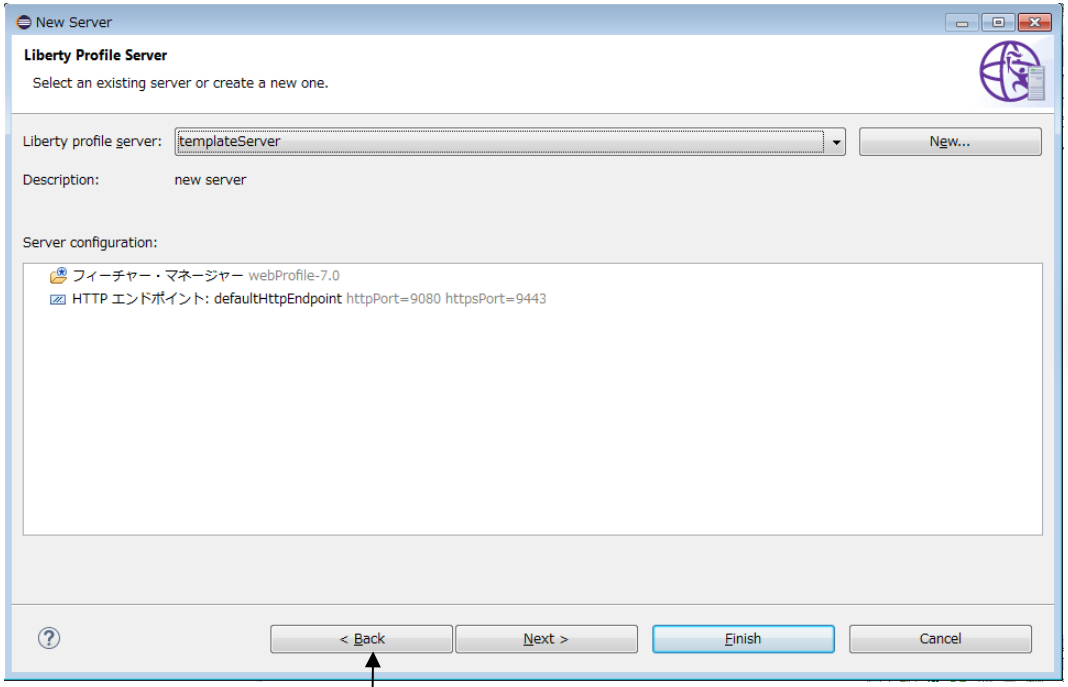


サーバーの追加を行います。

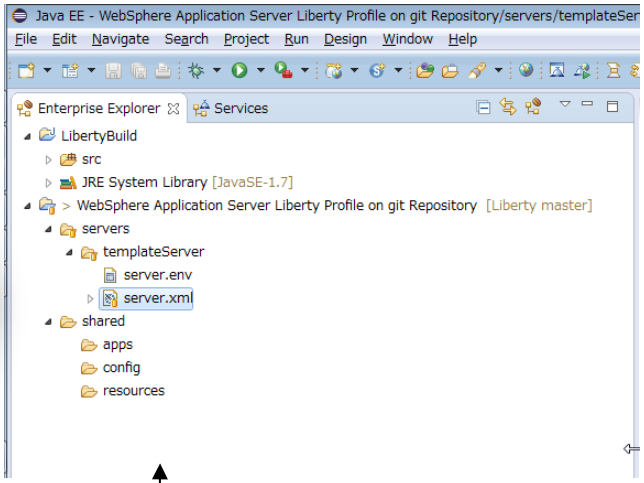


クローンされたLibertyサーバー環境を指定してサーバーの追加を行います。

4 ローカルリポジトリにクローン（eclipse プラグイン編）



作成しておいたtemplateServerを指定します。



無事サーバーの登録ができました。

サーバー変更を行った際には、server.xmlをadd、commit、PUSHしてリポジトリに反映する流れになります。

Maven

Tool：maven

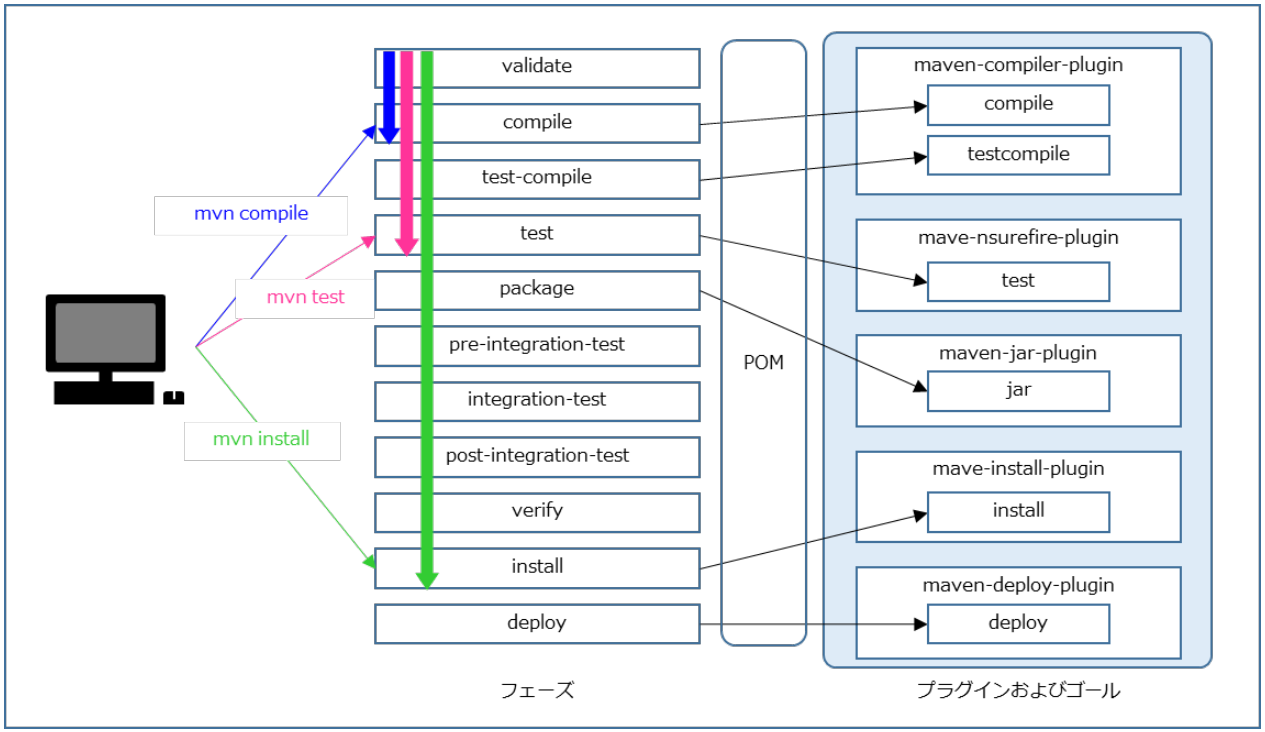
- ツール概要
  - 何ができるのか？
  - どこで使うか？
  - Libertyとの連携は？
- 使用手順
  - インストール
  - 初期セットアップ
  - 実行

ツール概要

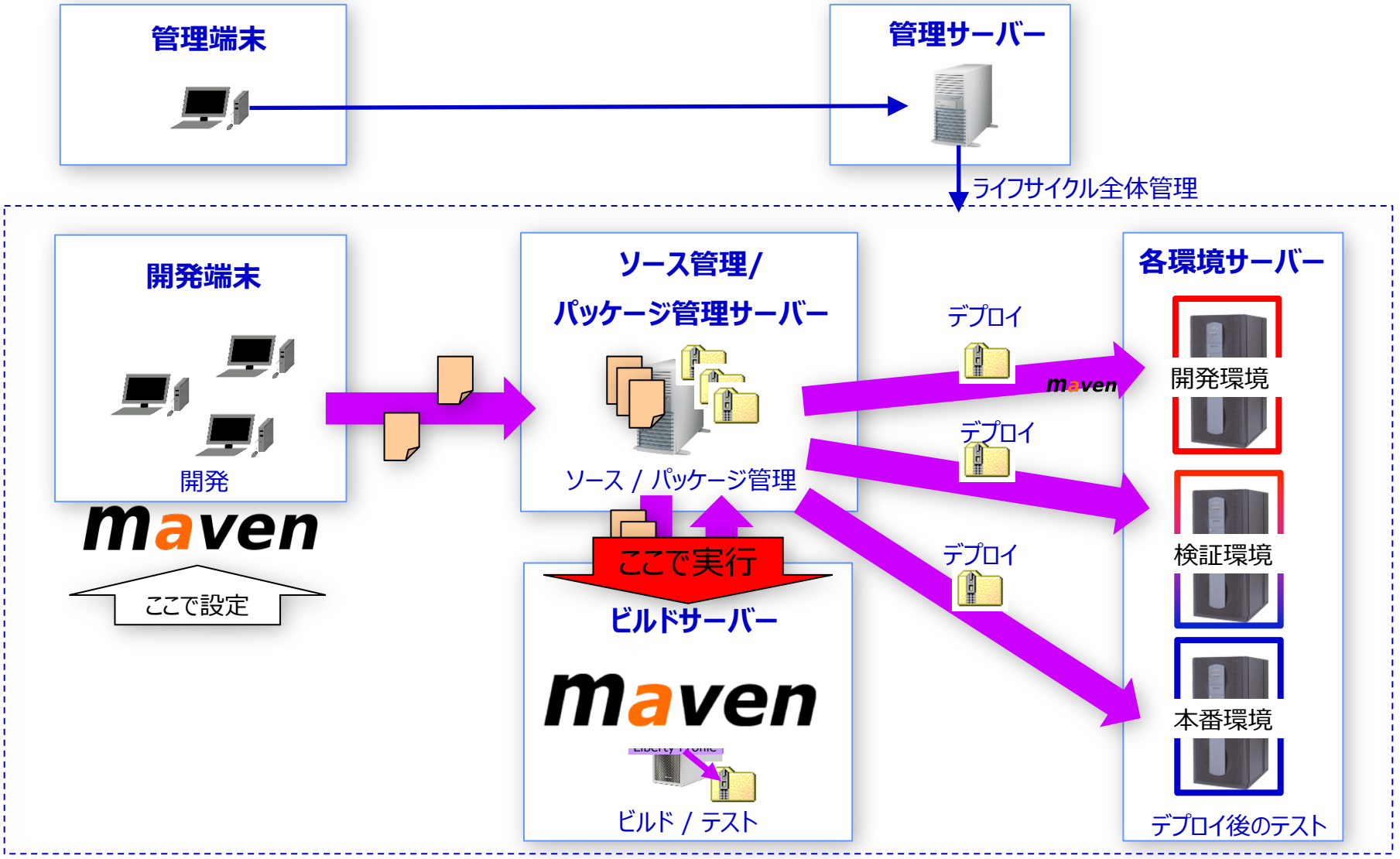
1. 何ができるのか？
2. どこで使うか？
3. Libertyとの連携は？

1. 何ができるのか？

- プロジェクトのライフサイクル全体を管理するソフトウェア・プロジェクト管理ツール
  - プロジェクト（アプリケーション）情報の一元管理
  - 依存ライブラリの管理
  - ビルドプロセスの単純化、自動化
    - プロジェクトのコンパイル、ユニットテスト



2. どこで使うか？



3. Libertyとの連携は？

- Liberty Mavenプラグインが提供されている
- Libertyサーバーの作成・開始・停止・アプリデプロイ・パッケージングを行うことが可能
- Mavenゴールとして以下が提供されている

Mavenゴール	説明
liberty:install-server	Libertyプロファイルのランタイムを導入する
liberty:create-server	Libertyサーバーを作成する
liberty:start-server	Libertyサーバーを起動する
liberty:stop-server	Libertyサーバーを停止する
liberty:package-server	Libertyサーバーをパッケージングする
liberty:dump-server	サーバーの診断情報をアーカイブに出力する
liberty:java-dump-server	サーバーJVMの診断情報を出力する
liberty:deploy	アプリケーションをLibertyサーバーのdropinsディレクトリーにデプロイする
liberty:undeploy	アプリケーションをLibertyサーバーのdropinsディレクトリーから削除する
liberty:install-feature	Esa（Subsystem Archive）形式でパッケージされたfeatureを導入する
liberty:uninstall-feature	Libertyプロファイルのランタイムからfeatureを削除する
liberty:install-apps	Mavenのdependenciesとして指定された１つ以上のアプリケーションをLibertyサーバーのdropinsディレクトリーにコピーする

https://github.com/WASdev/ci.maven?cm_mc_uid=54428987039714449566809&cm_mc_sid_50200000=1445860688#build

3. Libertyプロファイル管理用のマルチモジュール・プロジェクト

- また、Libertyプロファイル用のマルチモジュールプロジェクト生成のarchetypeとして、**liberty-plugin-archetype** が提供されています。
- このプロジェクトでは、
  - シンプルなwebアプリケーションのビルド
  - Libertyプロファイルサーバーへのデプロイ
  - Libertyプロファイルサーバー上でのテスト

  を行います。
  - また、そのアプリケーションを含むLibertyプロファイルのサーバーパッケージの作成も行います。

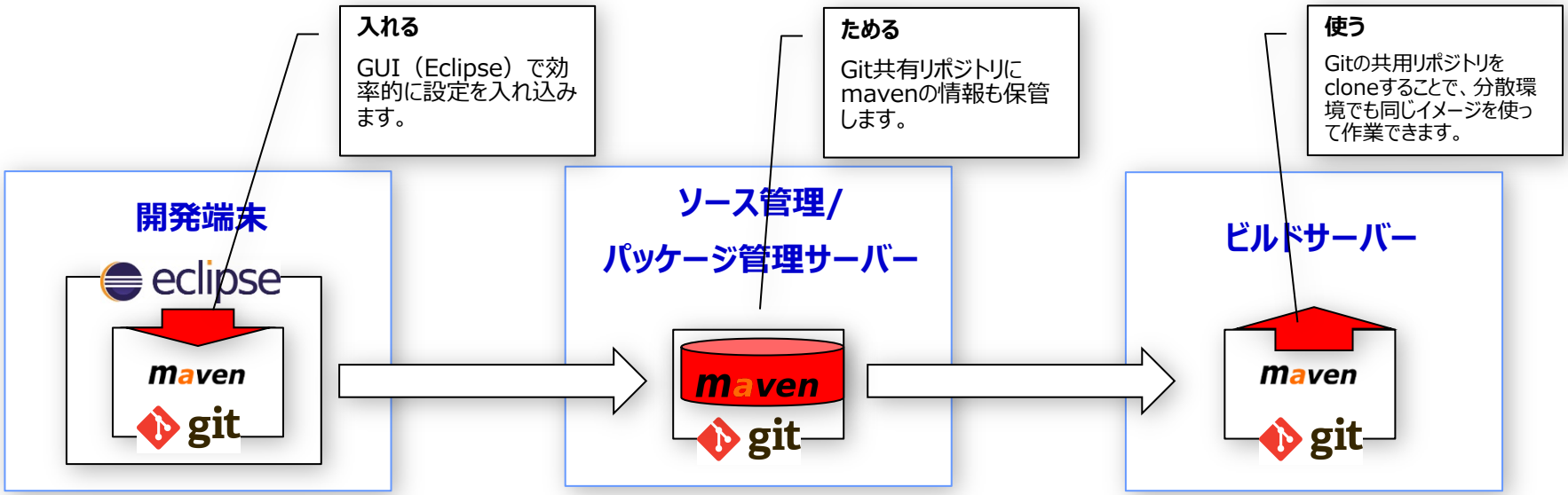
- 以降で、このプロジェクトのセットアップの手順も紹介します。

使用手順

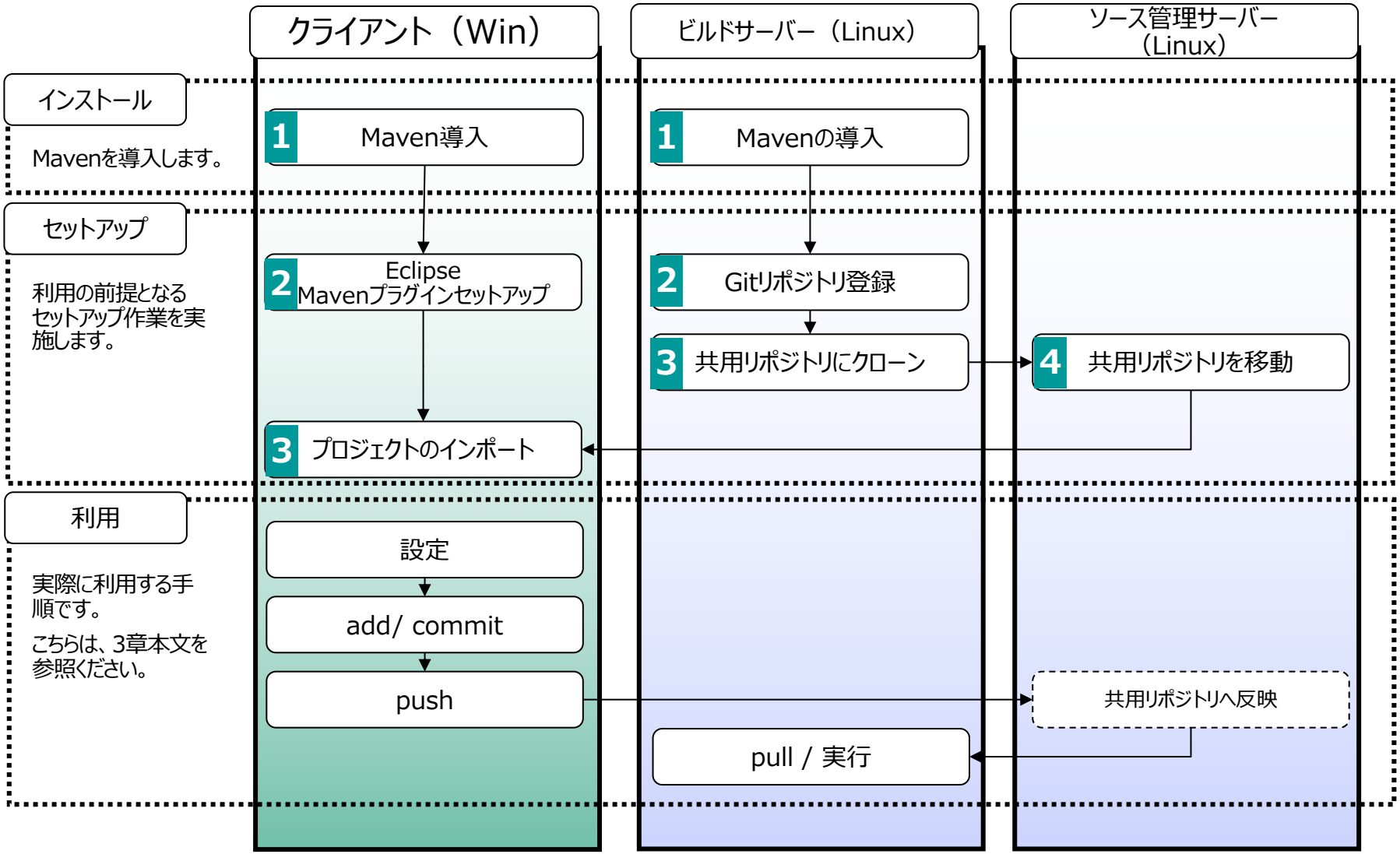
1. 前提
2. 全体の流れ
3. ビルドサーバー
    1. インストール
    2. 初期セットアップ
4. 開発端末
    1. インストール
    2. 初期セットアップ
5. 実行

1. 前提

- 今回のシステムではgit も使用していますので、Maven のリポジトリもgit リポジトリとして登録を行う事とします。
- 開発端末のEclipseでMavenの設定などを行い、git共用リポジトリ@SCMサーバーに登録し、buildサーバーでのそのプロジェクトを読み込んでビルドを行う、という分散管理の利点を生かす環境を構築します。



2. 全体の流れ



3.1. インストール @ビルドサーバー

Maven

1. Mavenの導入

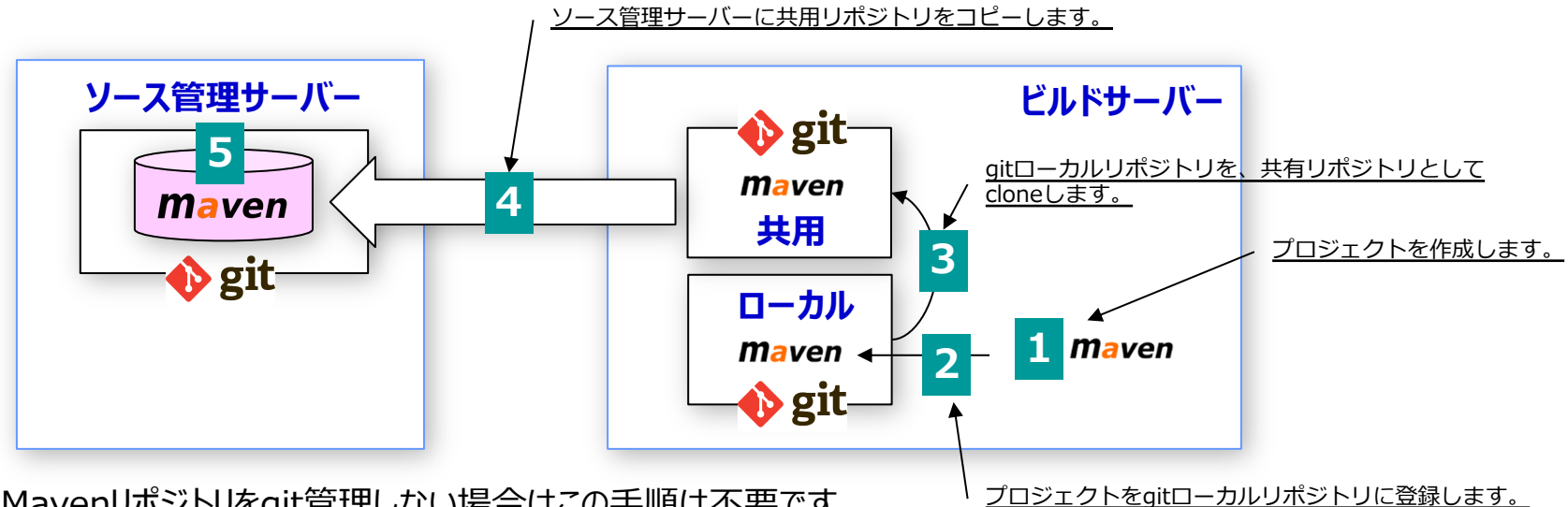
1. Mavenのモジュール(Binary zip)をダウンロードします。
   Apache Maven Project : Downloading Apache Maven 3.3.3
   http://maven.apache.org/download.cgi
2. ダウンロードしたzipファイルを任意のディレクトリーに解凍して導入は完了です。

2. システム環境変数の設定

変数名	変数値(例)
JAVA_HOME	C:¥Program Files¥Java¥jdk1.7.0_67
M2_HOME	C:¥IBM¥Maven323¥apache-maven-3.2.3
Path(既存)	%M2_HOME%¥bin;%JAVA_HOME%¥bin; (追記する)

3.2. 初期セットアップ @ ビルドサーバー (1/4)

- Mavenリポジトリとして使用する、gitの公開リポジトリを作成します。
- 以下の流れで作業します。
  1. ビルドサーバー上で、Mavenプロジェクトを作成します。
  2. ビルドサーバー上で、Mavenのリポジトリをgit登録します。これによってローカルのgitリポジトリが作成されます。
  3. ビルドサーバー上のローカルgitリポジトリを、同じくビルドサーバー上に共有リポジトリとしてcloneします。
  4. （共用リソースは、ソース管理サーバーに集約するという観点から）ビルドサーバー上の共用gitリポジトリを、ソース管理サーバーにコピーします。
  5. これで、ソース管理サーバー上にMavenプロジェクトがgit共用リポジトリとして登録されることになります。これ以降、ビルドサーバーはソース管理サーバーに接続、PULLしてプロジェクトリソースを参照します。



※ Mavenリポジトリをgit管理しない場合はこの手順は不要です。

3.2. 初期セットアップ @ ビルドサーバー (2/4)

Maven

1

- まずは、ビルドサーバー上にmavenプロジェクトを作成します。以下、実行例です。

```
[wasadmin@build maven]$ mvn archetype:generate ¥
> -DarchetypeGroupId=net.wasdev.wlp.maven ¥
> -DarchetypeArtifactId=liberty-plugin-archetype ¥
> -DgroupId=ManageLiberty ¥
> -DartifactId=ManageLiberty ¥
> -Dversion=1.0-SNAPSHOT ¥
> -DwlpInstallDir=/opt/IBM/Liberty/build/wlp/
[INFO] Scanning for projects...
Downloading:
http://public.dhe.ibm.com/ibmdl/export/pub/software/websphere/wasdev/maven/repository/
org/apache/maven/plugins/maven-metadata.xml
Downloading:
http://public.dhe.ibm.com/ibmdl/export/pub/software/websphere/wasdev/maven/repository/
org/codehaus/mojo/maven-metadata.xml
Downloading:
http://public.dhe.ibm.com/ibmdl/export/pub/software/websphere/wasdev/maven/repository/
org/apache/maven/plugins/maven-archetype-plugin/maven-metadata.xml
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Build Liberty package 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:2.4:generate (default-cli) > generate-sources @
build >>>
[INFO]
[INFO] <<< maven-archetype-plugin:2.4:generate (default-cli) < generate-sources @
build <<<
[INFO]
[INFO] --- maven-archetype-plugin:2.4:generate (default-cli) @ build ---
[INFO] Generating project in Interactive mode
[INFO] Archetype [net.wasdev.wlp.maven:liberty-plugin-archetype:1.0] found in catalog
remote
[INFO] Using property: groupId = ManageLiberty
[INFO] Using property: artifactId = ManageLiberty
[INFO] Using property: version = 1.0-SNAPSHOT
[INFO] Using property: package = ManageLiberty
[INFO] Using property: wlpInstallDir = /opt/IBM/Liberty/build/wlp/
[INFO] Using property: wlpPluginVersion = 1.0
```

```
Confirm properties configuration:
groupId: ManageLiberty
artifactId: ManageLiberty
version: 1.0-SNAPSHOT
package: ManageLiberty
wlpInstallDir: /opt/IBM/Liberty/build/wlp/
wlpPluginVersion: 1.0
 Y: : Y
[INFO] ------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Archetype: liberty-plugin-
archetype:1.0
[INFO] ------------------------------------------------------------------------
[INFO] Parameter: groupId, Value: ManageLiberty
[INFO] Parameter: artifactId, Value: ManageLiberty
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: package, Value: ManageLiberty
[INFO] Parameter: packageInPathFormat, Value: ManageLiberty
[INFO] Parameter: package, Value: ManageLiberty
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: wlpInstallDir, Value: /opt/IBM/Liberty/build/wlp/
[INFO] Parameter: groupId, Value: ManageLiberty
[INFO] Parameter: artifactId, Value: ManageLiberty
[INFO] Parameter: wlpPluginVersion, Value: 1.0
[INFO] Parent element not overwritten in
/opt/IBM/Liberty/maven/ManageLiberty/ManageLiberty-web/pom.xml
[INFO] Parent element not overwritten in
/opt/IBM/Liberty/maven/ManageLiberty/ManageLiberty-test/pom.xml
[INFO] Parent element not overwritten in
/opt/IBM/Liberty/maven/ManageLiberty/ManageLiberty-assembly/pom.xml
[INFO] project created from Archetype in dir: /opt/IBM/Liberty/maven/ManageLiberty
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 38.435 s
[INFO] Finished at: 2015-10-27T14:52:57+09:00
[INFO] Final Memory: 13M/32M
[INFO] ------------------------------------------------------------------------
```

3.2. 初期セットアップ @ ビルドサーバー (3/4)

- Mavenリポジトリとして使用する、gitの公開リポジトリを作成します。

2

```
[wasadmin@build maven]$ pwd
/opt/IBM/Liberty/maven
[wasadmin@build maven]$ git init
Initialized empty Git repository in /opt/IBM/Liberty/maven/.git/
[wasadmin@build maven]$ git add .
[wasadmin@build maven]$ git commit -m "initial commit"
[master (root-commit) fa02b15] initial commit
 14 files changed, 495 insertions(+)
 create mode 100644 ManageLiberty/ManageLiberty-assembly/pom.xml
 create mode 100644 ManageLiberty/ManageLiberty-assembly/src/test/resources/wlp/server.xml
 create mode 100644 ManageLiberty/ManageLiberty-test/pom.xml
 create mode 100644 ManageLiberty/ManageLiberty-test/src/test/java/ManageLiberty/test/ServletTest.java
 create mode 100644 ManageLiberty/ManageLiberty-test/src/test/resources/wlp/server.xml
 create mode 100644 ManageLiberty/ManageLiberty-web/pom.xml
 create mode 100644 ManageLiberty/ManageLiberty-web/src/main/java/ManageLiberty/web/SimpleServlet.java
 create mode 100644 ManageLiberty/ManageLiberty-web/src/main/webapp/index.html
 create mode 100644 ManageLiberty/ManageLiberty-web/target/ManageLiberty-web-1.0-SNAPSHOT.war
 create mode 100644 ManageLiberty/ManageLiberty-web/target/ManageLiberty-web-1.0-SNAPSHOT/WEB-
INF/classes/ManageLiberty/web/SimpleServlet.class
 create mode 100644 ManageLiberty/ManageLiberty-web/target/ManageLiberty-web-1.0-SNAPSHOT/index.html
 create mode 100644 ManageLiberty/ManageLiberty-web/target/classes/ManageLiberty/web/SimpleServlet.class
 create mode 100644 ManageLiberty/pom.xml
 create mode 100755 pom.xml
```

前頁で作成したMavenのリポジトリを、gitのローカルリポジトリとして登録しています。

3

```
[wasadmin@build maven]$ git clone --bare /opt/IBM/Liberty/maven/ /opt/IBM/Liberty/maven-shared/
Cloning into bare repository '/opt/IBM/Liberty/maven-shared'...
done.
```

Gitのローカルリポジトリを、共用リポジトリとしてcloneしています。

※ Mavenリポジトリをgit管理しない場合はこの手順は不要です。

3.2. 初期セットアップ @ ビルドサーバー (4/4)

- Mavenのgit共用リポジトリを、SCMサーバーに移動

4

```
[wasadmin@build maven-shared]$ scp -pr /opt/IBM/Liberty/maven-shared/ wasadmin@scm.ise.com:/opt/IBM/Liberty/
wasadmin@scm.ise.com's password:
HEAD                                                                    100%   23     0.0KB/s   00:00
config                                                                  100%  115     0.1KB/s   00:00
 :
c453e40a75e97db69050c6032e30ef0360c968                                  100%   55     0.1KB/s   00:00
description                                                             100%   73     0.1KB/s   00:00
[wasadmin@build maven-shared]$
```

- ビルド管理サーバーでは、このソース管理サーバー上の共用リポジトリに対して PUSH/PULLを行います。（PUSH/PULLなど詳細はgitの資料を参照ください。）

5

```
[wasadmin@build maven]$  git push --set-upstream ssh://wasadmin@oms94.ise.com/opt/IBM/Liberty/maven-shared master
wasadmin@oms94.ise.com's password:
Counting objects: 4, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 443 bytes | 0 bytes/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To ssh://wasadmin@scm.ise.com/opt/IBM/Liberty/maven-shared
   fa02b15..8ddfffb  master -> master
Branch master set up to track remote branch master from ssh://wasadmin@oms94.ise.com/opt/IBM/Liberty/maven-shared.
```

共用リポジトリへの情報の送信（PUSH）

```
[wasadmin@build maven]$ git pull
wasadmin@scm.ise.com's password:
From ssh://scm.ise.com/opt/IBM/Liberty/maven-shared
 * branch            master     -> FETCH_HEAD
Already up-to-date.
```

共用リポジトリからの情報の取得（PULL）

※ Mavenリポジトリをgit管理しない場合はこの手順は不要です。

4.1. インストール@開発端末

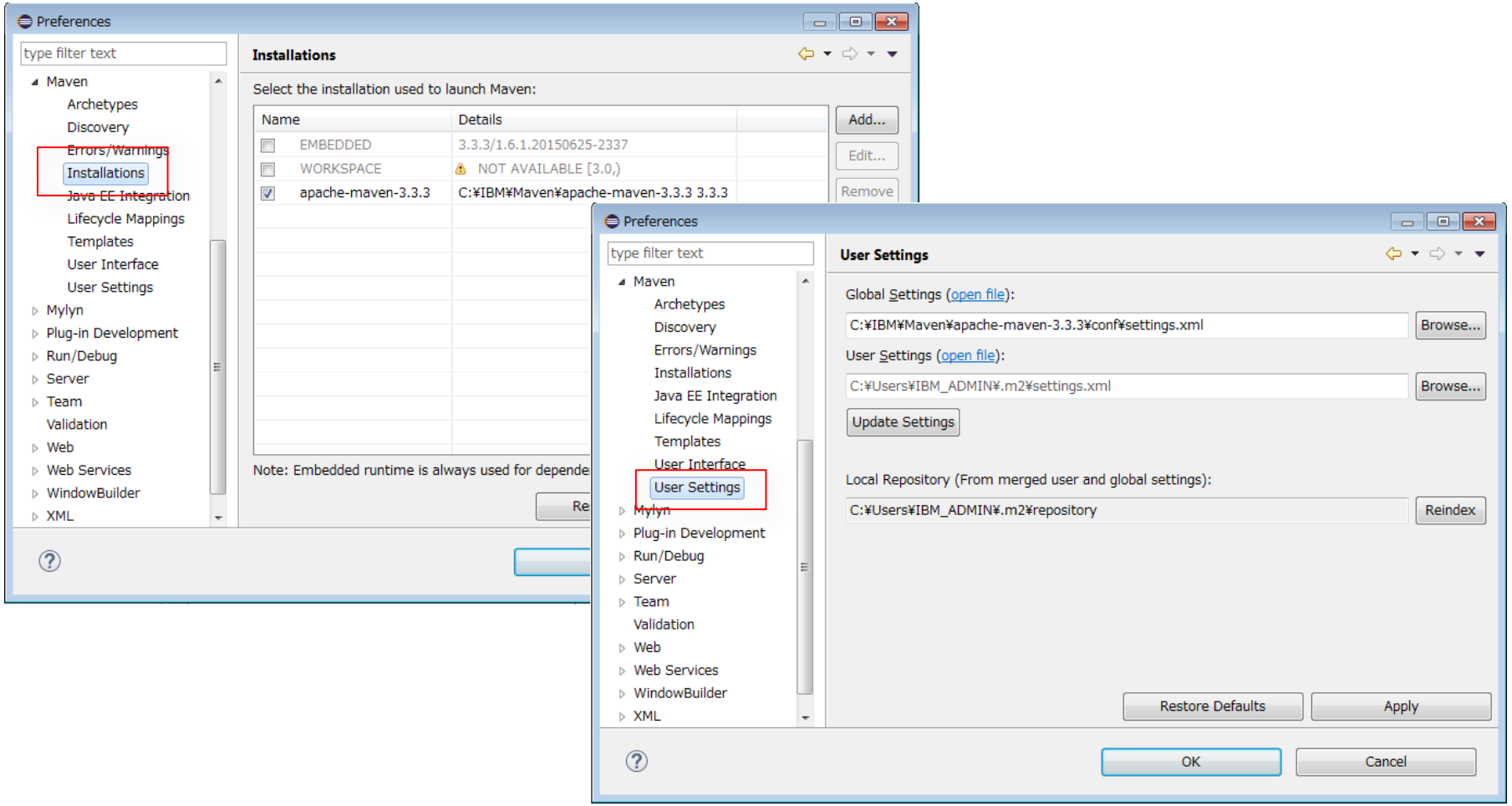
Maven

1. Mavenの導入
   1. Mavenのモジュール(Binary zip)をダウンロードする
      Apache Maven Project : Downloading Apache Maven 3.3.3
      http://maven.apache.org/download.cgi
   2. ダウンロードしたzipファイルを任意のディレクトリーに解凍して導入は完了
2. JDKの導入
3. Libertyの導入
4. システム環境変数の設定

変数名	変数値(例)
JAVA_HOME	C:¥Program Files¥Java¥jdk1.7.0_67
M2_HOME	C:¥IBM¥Maven323¥apache-maven-3.2.3
Path(既存)	%M2_HOME%¥bin;%JAVA_HOME%¥bin; (追記する)

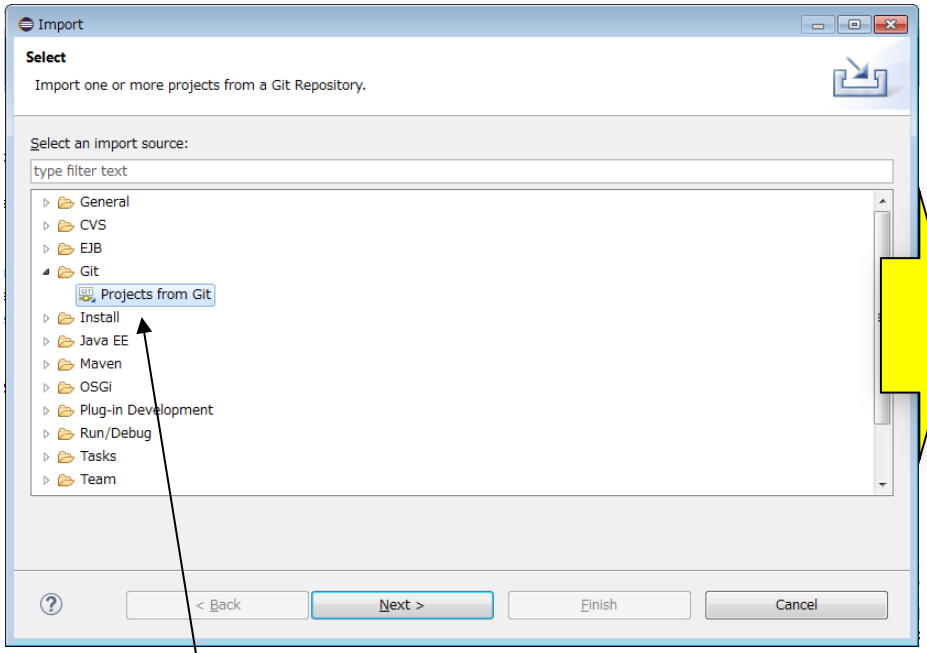
4.2. 初期セットアップ @ 開発端末 (1/4)

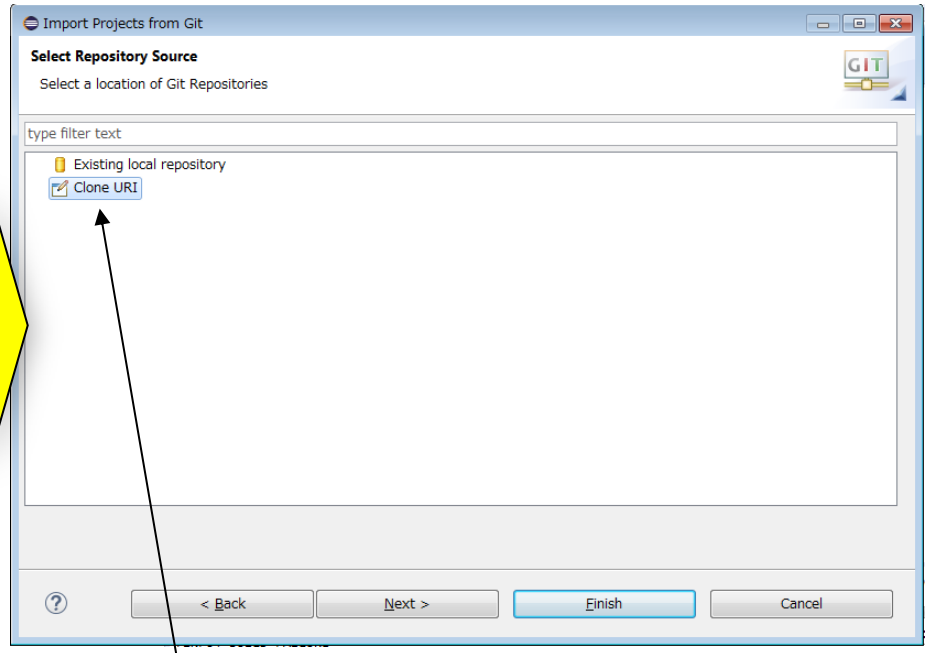
- Eclipseの設定
  - Window ⇒ preferences から、以下の箇所の設定を行います。



4.2. 初期セットアップ @ 開発端末 (2/4)

- 「3.2」で登録した、gitの共用リポジトリに登録されているMavenプロジェクトをインポートします。



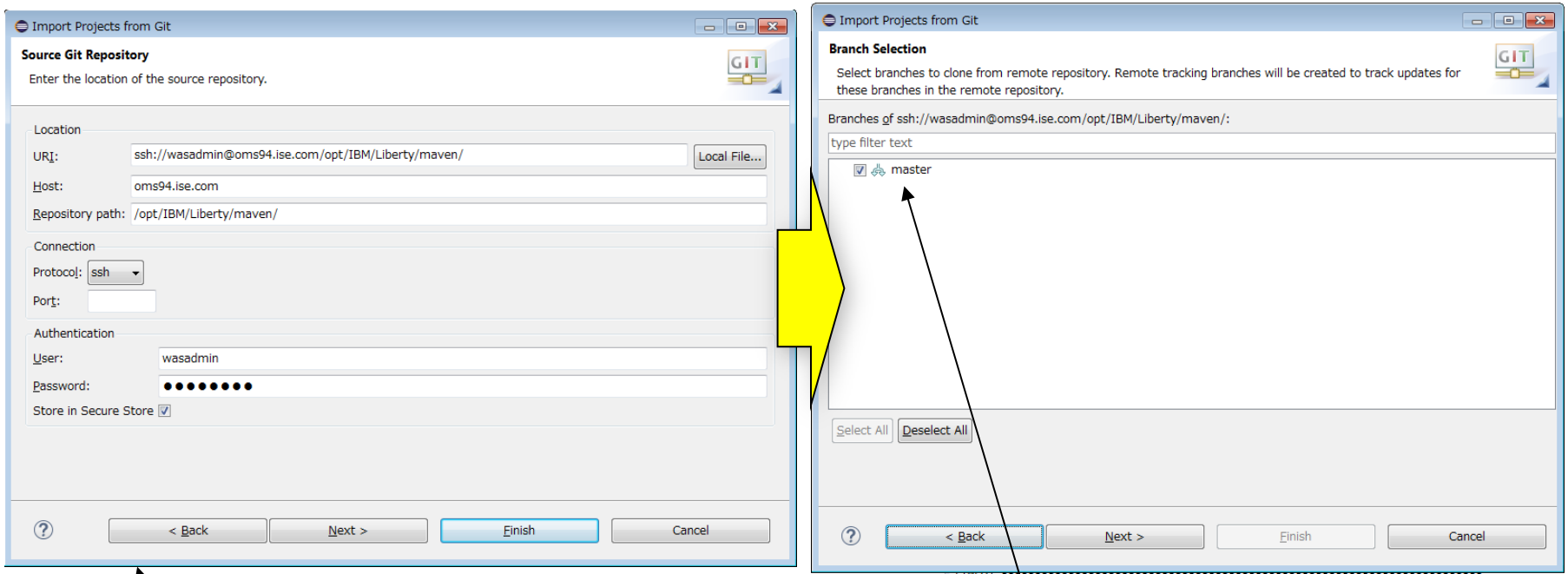


File ⇒ Import から開いたwindowで、
git ⇒ Projects from Git を選択します。

Clone URIを選択して、共用リポジトリからローカルリポジトリをコピー作成します。

4.2. 初期セットアップ @ 開発端末 (3/4)

- 続きです。

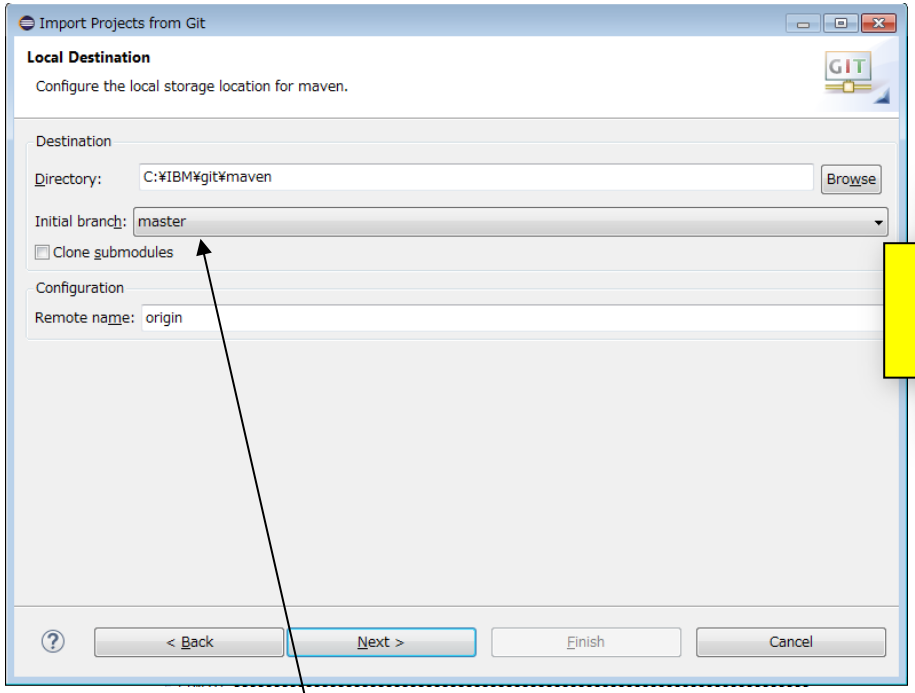


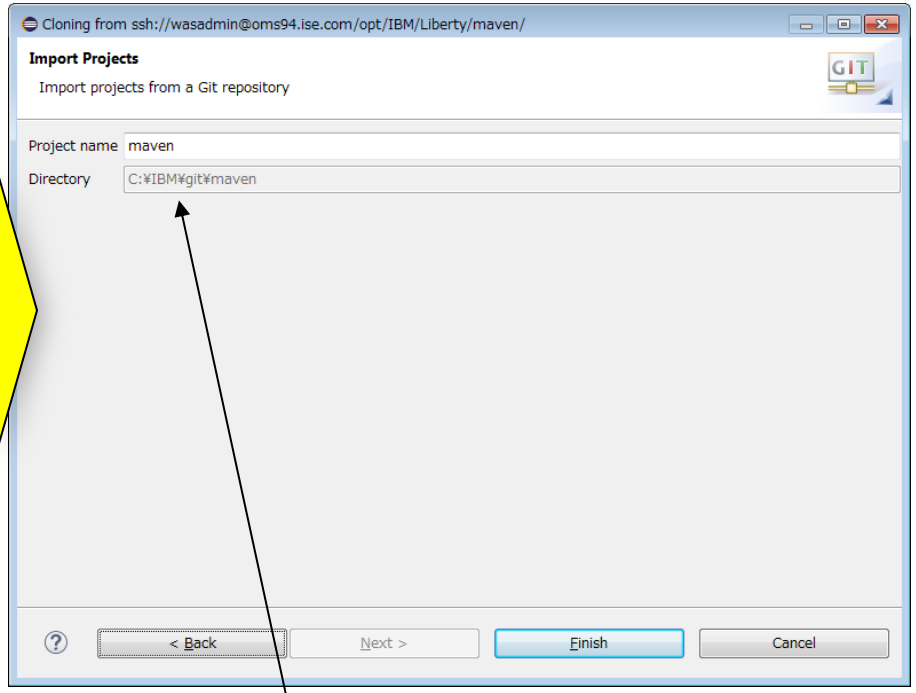
URL等必要な情報を入力します。

ブランチを選択します。

4.2. 初期セットアップ @ 開発端末 (4/4)

- またまた続きです。





コピー先のディレクトリを指定します。

プロジェクト名を指定して完了です！

5. 実行 - プラグインの有効化

- Liberty Mavenプラグインを使用可能にする
  - pom.xmlに設定を追加する

```
<project>
   ...
   <build>
     <finalName>MvnWebHello</finalName>
       <plugins>
          <plugin>
             <groupId>net.wasdev.wlp.maven.plugins</groupId>
             <artifactId>liberty-maven-plugin</artifactId>
             <version>1.0</version>
          </plugin>
       </plugins>
   </build>
</project>
```

※ 3.2. 初期セットアップ @ ビルドサーバー (2/4) の手順でプロジェクトを作成した場合、
プラグイン指定はpom.xmlに既に含まれています。

5. 実行 - 利用方法

- 利用方法は２通り
  - Liberty Mavenプラグインを単体で利用する
    - mvnコマンドにLibertyのMavenゴールを指定することでLibertyサーバーの作成・起動・停止などを行う
  - Mavenライフサイクルの中でLiberty Mavenプラグインを利用する
    - pom.xmlにLiberty Mavenプラグインを実行したいフェーズと実行するゴールを記述することにより、プラグインをビルドライフサイクル中の任意のタイミングで実行する

例）統合テストフェーズでの利用

フェーズ	ゴール	処理内容
pre-integration-test	start-server	Libertyサーバーを起動
	deploy	Libertyサーバーにアプリケーションをデプロイ
post-integration-test	stop-server	Libertyサーバーを停止
	package-server	Libertyサーバーをパッケージング

5. 実行 - Liberty Mavenプラグインを単体で利用する場合(1/2)

- Libertyサーバーの作成
  - プロジェクトのpom.xmlが存在するディレクトリで以下のコマンドを実行する

```
mvn liberty:create-server -DserverHome=<Liberty導入ディレクトリ> -DserverName=<サーバー名>
```

- Libertyサーバーの起動

```
mvn liberty:start-server -DserverHome=<Liberty導入ディレクトリ> -DserverName=<サーバー名>
```

- Libertyサーバーの停止

```
mvn liberty:stop-server -DserverHome=<Liberty導入ディレクトリ> -DserverName=<サーバー名>
```

- Libertyサーバーのパッケージング

```
mvn liberty:package-server -DserverHome=<Liberty導入ディレクトリ> -DserverName=<サーバー名>
                           -DpackageFile=<パッケージングファイル>
```

- アプリケーションのデプロイ

```
mvn liberty:deploy -DserverHome=<Liberty導入ディレクトリ> -DserverName=<サーバー名>
                   -DappArchive=<アプリケーションモジュール>
```

- アプリケーションの削除

```
mvn liberty:undeploy -DserverHome=<Liberty導入ディレクトリ> -DserverName=<サーバー名>
                     -DappArchive=<アプリケーションモジュール>
```

5. 実行 - Liberty Mavenプラグインを単体で利用する場合(2/2)

- pom.xmlの<configration>タグに導入ディレクトリやサーバー名を指定しておくと、コマンド実行時に引数として指定する必要がなくなる

```
<plugin>
        <groupId>net.wasdev.wlp.maven.plugins</groupId>
        <artifactId>liberty-maven-plugin</artifactId>
        <version>1.0</version>
        <configuration>
            <installDirectory>C:¥IBM¥Liberty8553¥wlp</installDirectory>
            <serverName>LibertyServer01</serverName>
        </configuration>
</plugin>
```

- Libertyサーバーの起動

```
mvn liberty:start-server
```

5. 実行 - Mavenライフサイクルの中でMaven Libertyプラグインを利用する場合 (1/4)

- pre-integration-testフェーズで「start-server」と「deploy」を実行する場合、以下のようにpom.xmlを設定する

```
<plugin>
  <groupId>net.wasdev.wlp.maven.plugins</groupId>
  <artifactId>liberty-maven-plugin</artifactId>
  <version>1.0</version>
  <executions>
    <execution>
      <id>start-liberty-server</id>
      <phase>pre-integration-test</phase>
      <goals>
        <goal>start-server</goal>
      </goals>
    </execution>
```

（右に続く）

Libertyサーバーの起動

```
    <execution>
      <id>deploy web application</id>
      <phase>pre-integration-test</phase>
      <goals>
        <goal>deploy</goal>
      </goals>
      <configuration>
        <appArchive>
          ${project.build.directory}/MvnWebHello.war
        </appArchive>
      </configuration>
    </execution>
  </executions>
  <configuration>
    <installDirectory>C:¥IBM¥Liberty8553¥wlp</installDirectory>
    <serverName>LibertyServer01</serverName>
  </configuration>
</plugin>
```

アプリケーションのデプロイ

5. 実行 - Mavenライフサイクルの中でMaven Libertyプラグインを利用する場合 (2/4)

- 前項のpom.xmlを定義した後に以下のコマンドを実行する
  - 「pre-integration-test」までのフェーズ(compileやpackage等)が実行され、さらに「pre-integration-test」フェーズにて、Libertyサーバーの起動とアプリケーションのデプロイが実行される

```
mvn pre-integration-test
```

5. 実行 - Mavenライフサイクルの中でMaven Libertyプラグインを利用する場合 (3/4)

- post-integration-testフェーズで「stop-server」と「package-server」を実行する場合、以下のようにpom.xmlを設定する

```
<plugin>
  <groupId>net.wasdev.wlp.maven.plugins</groupId>
  <artifactId>liberty-maven-plugin</artifactId>
  <version>1.0</version>
  <executions>
    .....
    <execution>
      <id>stop-liberty-server</id>
      <phase>post-integration-test</phase>
      <goals>
        <goal>stop-server</goal>
      </goals>
    </execution>
```

（右に続く）

Libertyサーバーの停止

```
    <execution>
      <id>package liberty server</id>
      <phase>post-integration-test</phase>
      <goals>
        <goal>package-server</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <installDirectory>C:¥IBM¥Liberty8553¥wlp</installDirectory>
    <serverName>LibertyServer01</serverName>
  </configuration>
</plugin>
```

Libertyサーバーのパッケージング

5. 実行 - Mavenライフサイクルの中でMaven Libertyプラグインを利用する場合 (4/4)

- 前項のpom.xmlを定義した後に以下のコマンドを実行する
  - コマンドを実行すると、「post-integration-test」までのフェーズが実行され、さらに「post-integration-test」フェーズにて、Libertyサーバーの停止、およびLibertyサーバーのパッケージ化が実行され、最後の「install」フェーズにてローカルリポジトリへの登録が実行される

```
mvn install
```

【参考】package-serverの実行例

- 以下は、このプロジェクトにおいて、Package-serverのみ単独で実行した場合の出力例です。

```
[wasadmin@build maven]$ mvn liberty:package-server -
DserverHome=/opt/IBM/Liberty/build/wlp/ -DserverName=templateServer
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Build Order:
[INFO]
[INFO] Build Liberty package
[INFO] Simple Web Application Parent
[INFO] Simple Web Application
[INFO] Simple Web Application Iteration Tests
[INFO] Simple Web Application Server Assembly
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Build Liberty package 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------

 :

[INFO] Packaging server templateServer.
[INFO] Server templateServer package complete in
/opt/IBM/Liberty/build/wlp/usr/servers/templateServer/templateServer.zip.
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Simple Web Application Parent 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- liberty-maven-plugin:1.0:package-server (default-cli) @ ManageLiberty-
parent ---

 :

[INFO] Packaging server templateServer.
[INFO] Server templateServer package complete in
/opt/IBM/Liberty/build/wlp/usr/servers/templateServer/templateServer.zip.
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Simple Web Application 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------

 :
```

```
[INFO] Packaging server templateServer.
[INFO] Server templateServer package complete in
/opt/IBM/Liberty/build/wlp/usr/servers/templateServer/templateServer.zip.
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Simple Web Application Iteration Tests 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]

 :

[INFO] Packaging server templateServer.
[INFO] Server templateServer package complete in
/opt/IBM/Liberty/build/wlp/usr/servers/templateServer/templateServer.zip.
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Simple Web Application Server Assembly 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]

 :

[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] Build Liberty package ............................. SUCCESS [ 24.939 s]
[INFO] Simple Web Application Parent ..................... SUCCESS [ 16.331 s]
[INFO] Simple Web Application ............................ SUCCESS [ 15.563 s]
[INFO] Simple Web Application Iteration Tests ............ SUCCESS [ 18.389 s]
[INFO] Simple Web Application Server Assembly ............ SUCCESS [ 18.695 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 01:34 min
[INFO] Finished at: 2015-10-27T15:38:26+09:00
[INFO] Final Memory: 6M/11M
[INFO] ------------------------------------------------------------------------
```

Jenkins

Jenkins

Tool：Jenkins

Jenkins

- ツール概要
  - 何ができるのか？
  - どこで使うか？
  - Libertyとの連携は？
  - 仕組み / アーキテクチャ
- 使用手順
  - インストール
  - セットアップ

ツール概要

1. 何ができるのか？
2. どこで使うか？
3. Libertyとの連携は？
4. 仕組み / アーキテクチャ

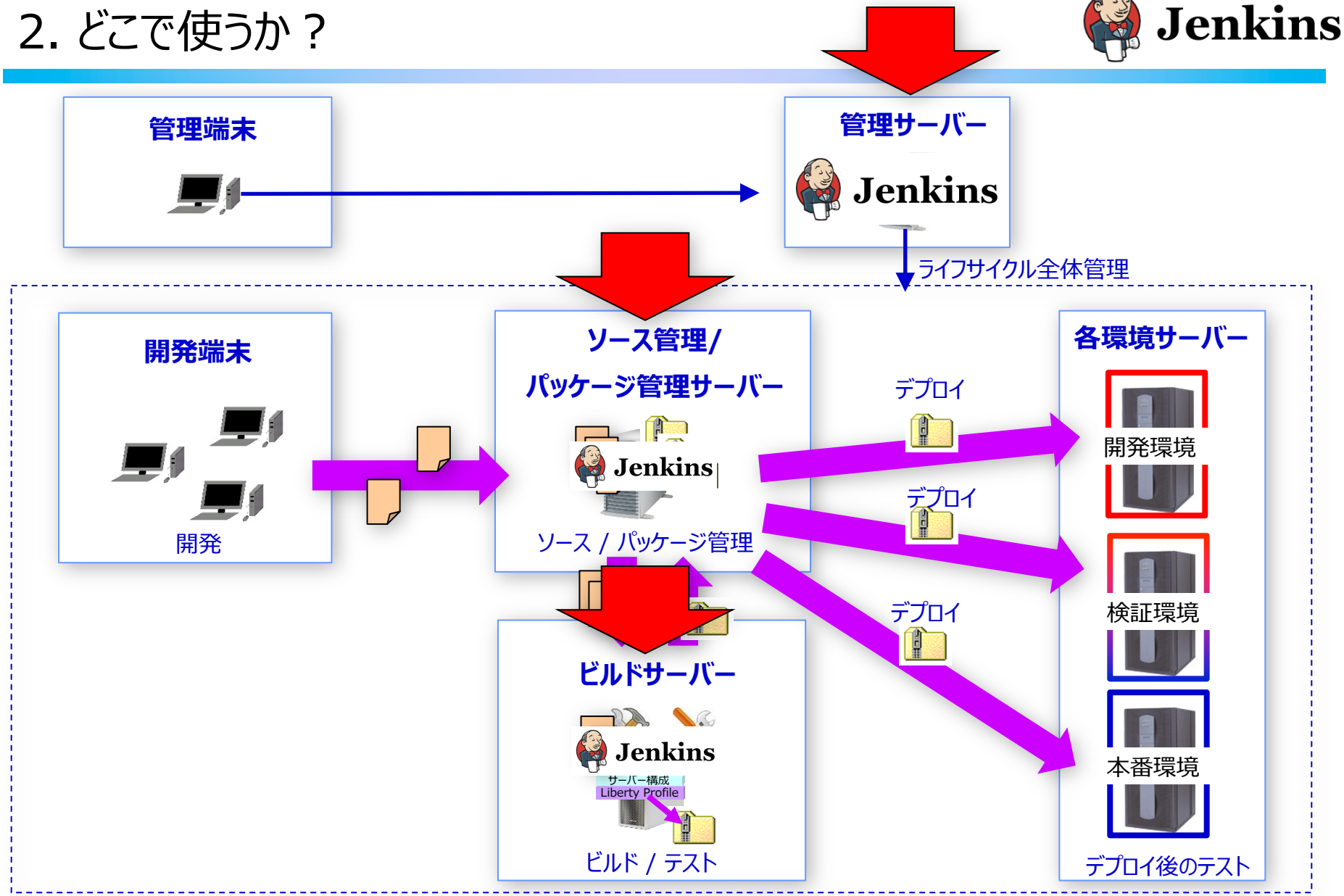
1. 何ができるのか？

Jenkins

- Jenkinsとは
  - Javaで書かれたオープンソースの継続的インテグレーションツール
    - ※ 継続的インテグレーションツール
      - アプリケーション作成時の品質改善や納期短縮を目的としてビルドやテストなどを継続的な実行を支援するツール
  - 反復型ジョブを実行する自動化フレームワーク
  - Hudsonプロジェクトが開発していたもののフォーク

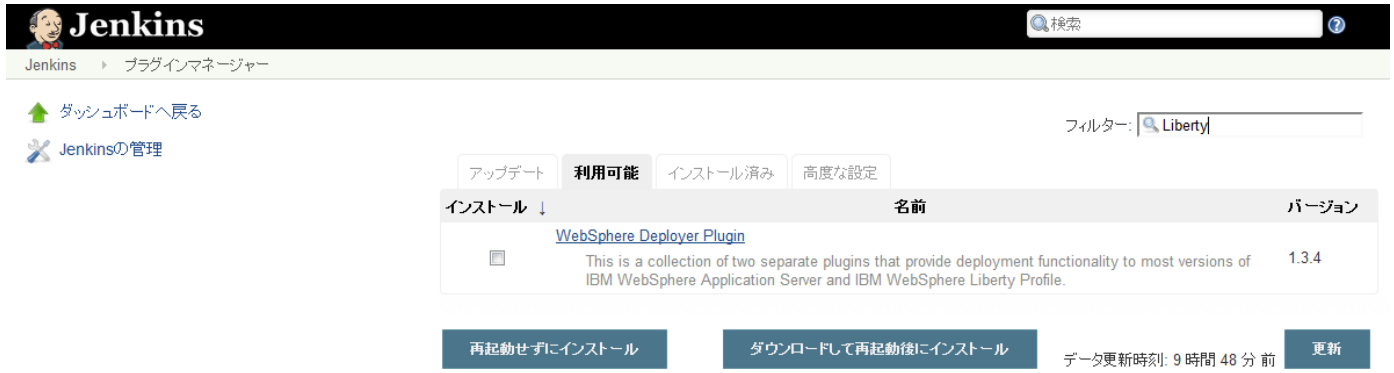
  - 要は、予め定義しておいた作業（ビルドやシェルの実行など）を、分散環境で、ブラウザからの簡単操作で自動実行してくれる、というツールです。

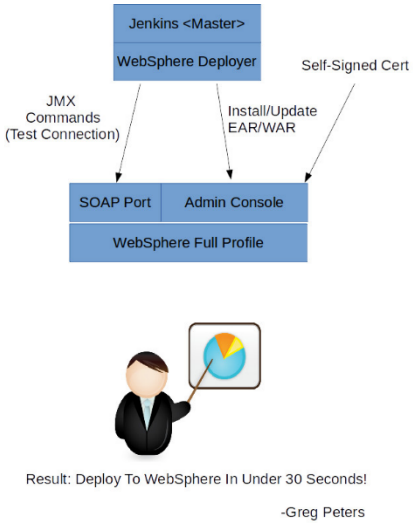
2. どこで使うか？



3. Libertyの連携は？

- 以下の通り、アプリケーションのデプロイ用プラグインが提供されています。





Plugin Features

•Deployment to a running remote or local instance of WebSphere 6.0.x - 8.5.x

•Deployment to a running remote or local instance of WebSphere Liberty Edition 8.x.x

•Support for deployments when SSL is activated on WebSphere (Global Security)

•Support for most versions of WebSphere (Liberty,ND,Extreme Scale, Express, Standard, etc...)

•Support for deploying to different WebSphere instances/types per build

•Support for node, cell, and server level deployments

•Support for pre-compiling JSPs on WebSphere during deployment (When stress testing as part of build process)

•Support for disabling JSP reloading (When stress testing as part of build process)

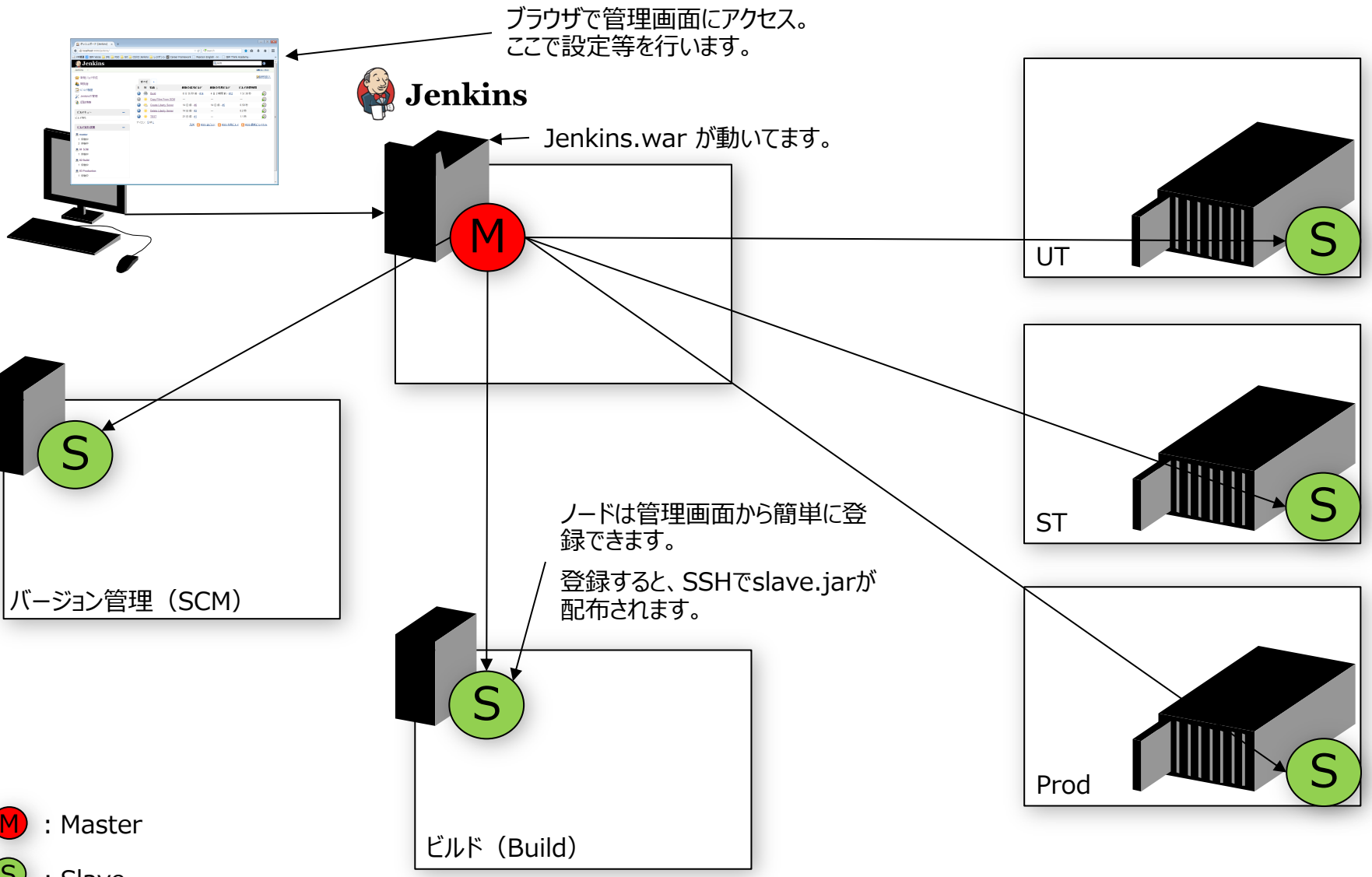
•Support for not starting application after deployment

•Supports fully stopping, uninstalling, installing and starting application during deployment process

•Smart detection of existing deployed application state (to prevent failures when a system admin stops your application without you knowing)

https://wiki.jenkins-ci.org/display/JENKINS/WebSphere+Deployer+Plugin

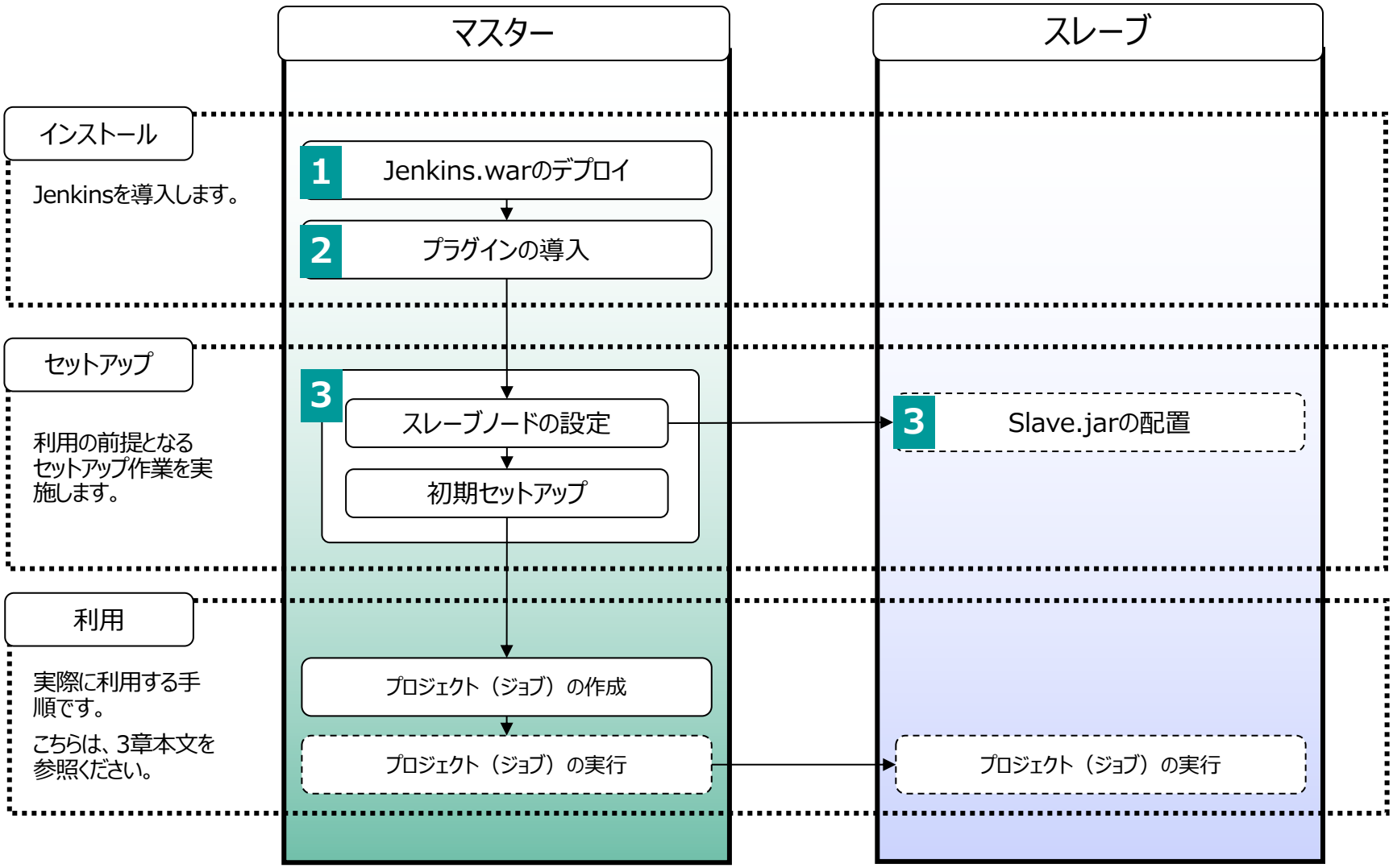
4. 仕組み/アーキテクチャー



使用手順

1. 全体の流れ
2. インストール
3. 初期セットアップ

1. 全体の流れ



2. インストール

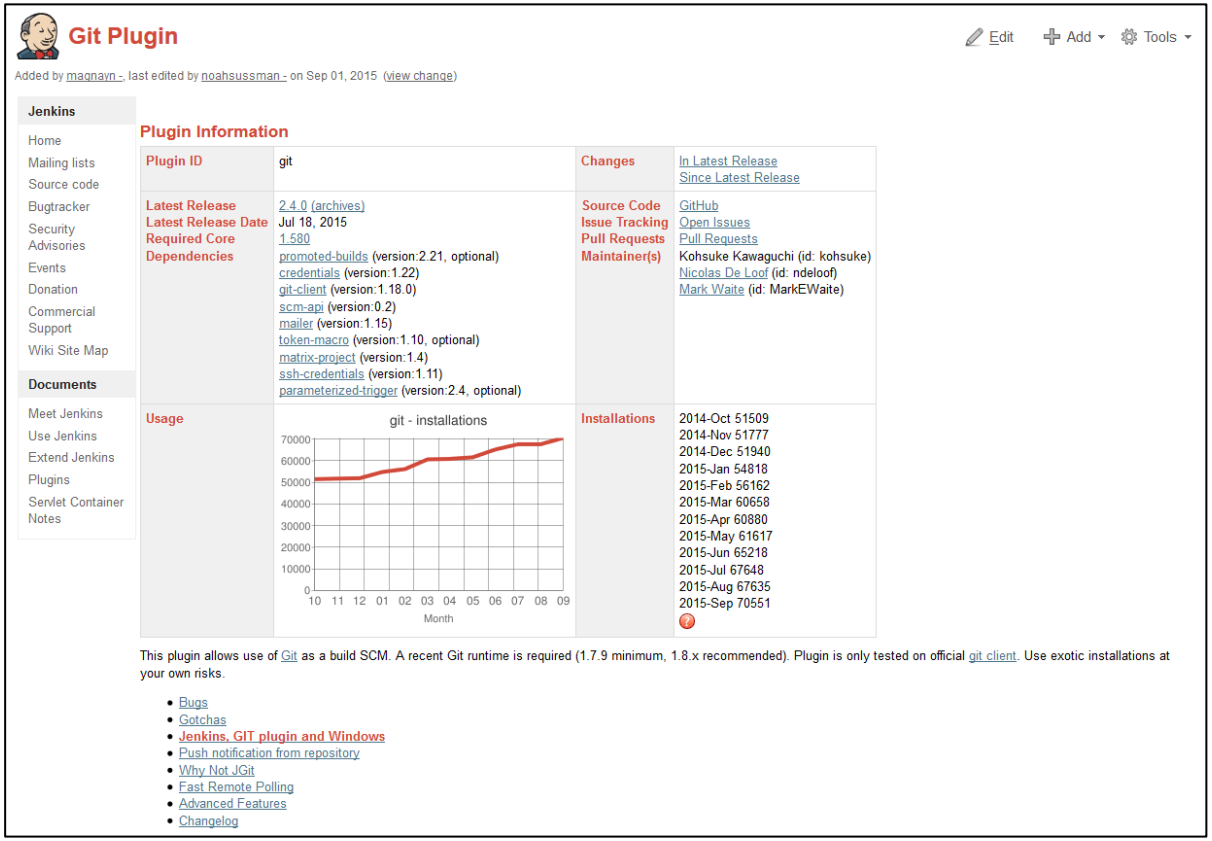
- マスター（サーバー）
  - いくつか方法はありますが・・・
    - 今回はLibertyを活用した方法ということで、warファイルをダウンロードしてデプロイする方式で導入します。

1. jenkins.warをダウンロードします。
   - 以下からダウンロードできます。
     - http://jenkins-ci.org/

2. 任意のAPサーバーにデプロイします。以下の手順を実施します。
   1. Libertyサーバーを作成します
   2. サーバー起動
   3. Jenkins.warをdropinsへ配置します。
   4. http://hostname:port/jenkins でアクセスできてしまいます。

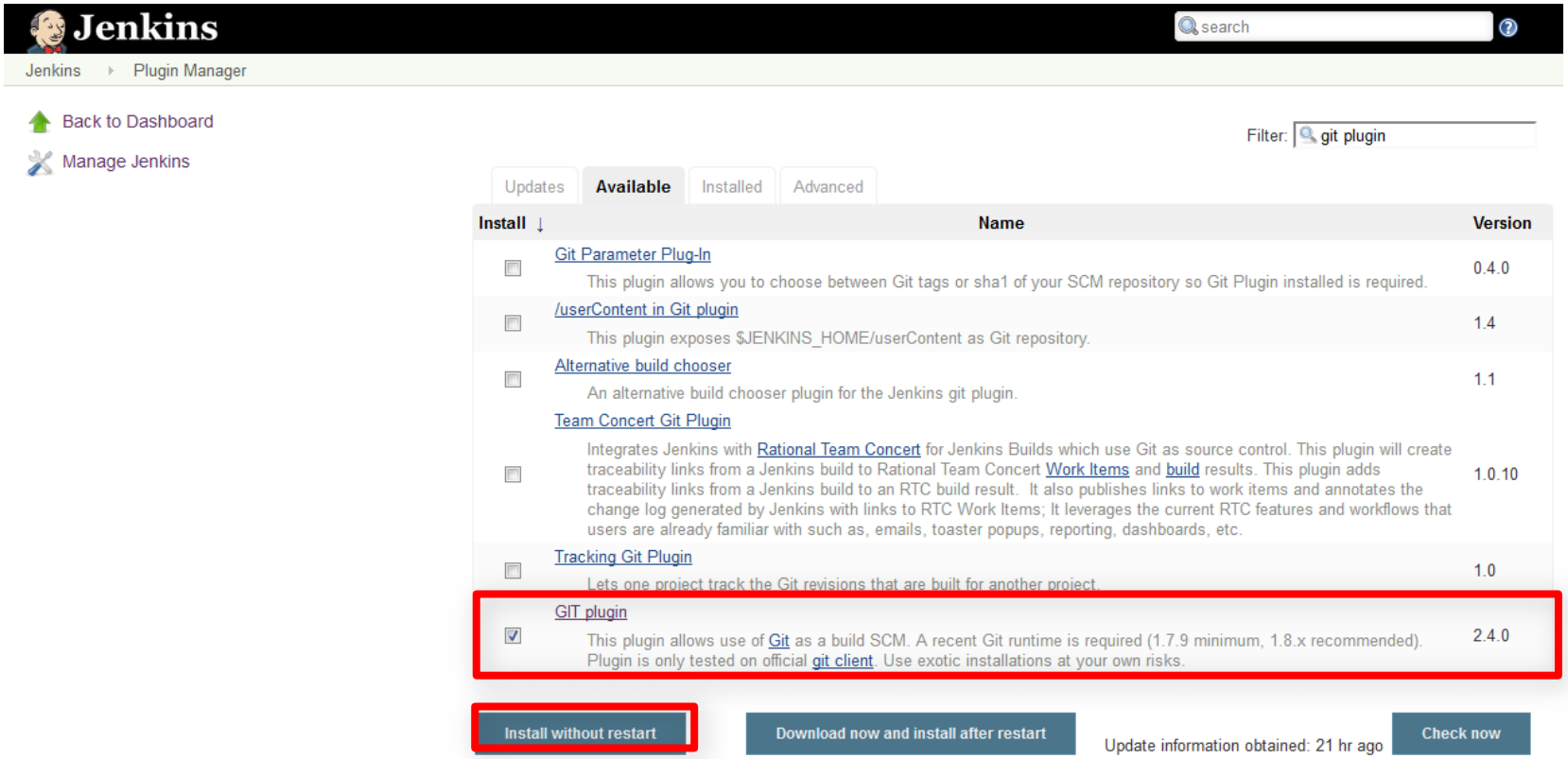
2. インストール

- gitプラグイン の導入
  - 管理画面から検索・表示できます。以下は、git pluginの情報です。



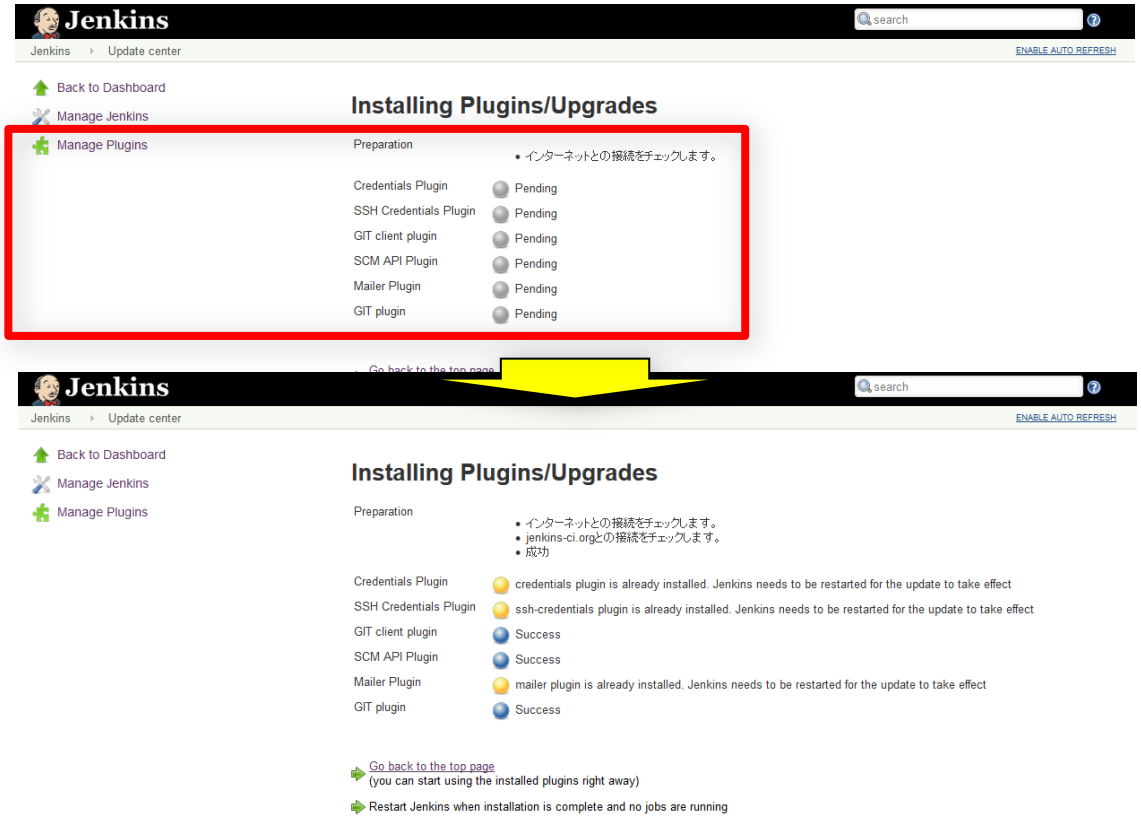
2. インストール

- gitプラグイン の導入
  - 一覧から選択して導入します。



2. インストール

- gitプラグイン の導入
  - 前提となるプラグインも一緒に導入されます。

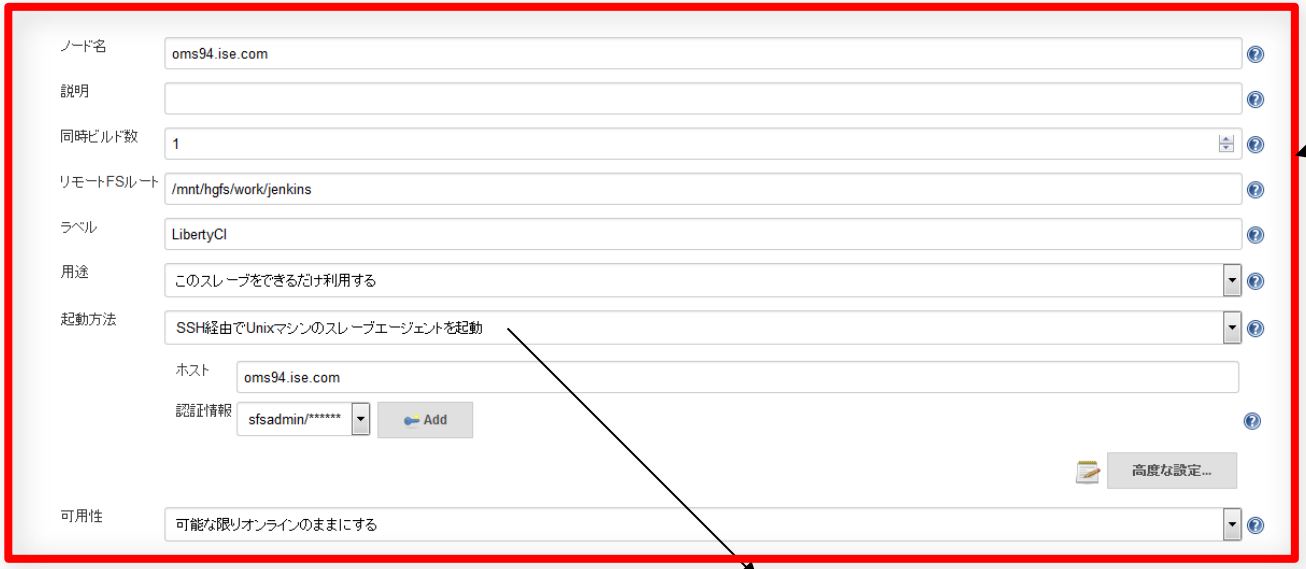


3. 初期セットアップ – スレーブノードの追加(1/4)

- スレーブ（エージェント）の追加
  - 管理画面から、導入先サーバーへのSSH接続設定をするだけ！

3. 初期セットアップ – スレーブノードの追加(2/4)

- スレーブ（エージェント）の追加



必要事項を入力します。

スレーブノードの起動方法を選択できます。

3. 初期セットアップ – スレーブノードの追加(3/4)

Jenkins

- スレーブ（エージェント）の起動時のコンソール出力例

```
[09/10/15 19:31:28] [SSH] oms94.ise.com:22とのSSHコネク

[09/10/15 19:31:29] [SSH] 認証成功
[09/10/15 19:31:29] [SSH] リモートユーザーの環境:
BASH=/bin/bash
BASHOPTS=cmdhist:extquote:force_fignore:hostcomplete:interactive_comments:progcomp:promptvars:sourcepath
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_EXECUTION_STRING=set
BASH_LINENO=()
BASH_SOURCE=()
BASH_VERSINFO=([0]="4" [1]="1" [2]="2" [3]="1" [4]="release" [5]="x86_64-redhat-linux-gnu")
BASH_VERSION='4.1.2(1)-release'
DIRSTACK=()
EUID=500
GROUPS=()
G_BROKEN_FILENAMES=1
HOME=/home/sfsadmin
HOSTNAME=oms94.ise.com
HOSTTYPE=x86_64
ID=500
IFS=$' ¥t¥n'
LANG=en_US.UTF-8
LESSOPEN='|/usr/bin/lesspipe.sh %s'
LOGNAME=sfsadmin
MACHTYPE=x86_64-redhat-linux-gnu
MAIL=/var/mail/sfsadmin
OPTERR=1
OPTIND=1
OSTYPE=linux-gnu
PATH=/usr/local/bin:/bin:/usr/bin
PIPESTATUS=([0]="0")
PPID=62290
PS4='+ '
[PWD=/home/sfsadmin
SELINUX_LEVEL_REQUESTED=
SELINUX_ROLE_REQUESTED=
SELINUX_USE_CURRENT_RANGE=
SHELL=/bin/bash
SHELLOPTS=braceexpand:hashall:interactive-comments
SHLVL=1
SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass
SSH_CLIENT='192.168.9.1 63712 22'
SSH_CONNECTION='192.168.9.1 63712 192.168.9.144 22'
TERM=dumb
UID=500
USER=sfsadmin
_=/etc/bashrc
__udisks ()
{
    local IFS='

[09/10/15 19:31:29] [SSH] sftpクライアントの開始
[09/10/15 19:31:29] [SSH] 最新のslave.jarをコピー中...
[09/10/15 19:31:29] [SSH] 489,023 バイトコピー.
Expanded the channel window size to 4MB
[09/10/15 19:31:29] [SSH] スレーブのプロセスを開始: cd "/mnt/hgfs/work/jenkins" && /opt/IBM/WebSphere/AppServer/java_1.7_64/jre/bin/java  -jar slave.jar

<===[JENKINS REMOTING CAPACITY]===>channel started

Slave.jar version: 2.52
これはUnixの

Evacuated stdout

Slave successfully connected and online
```

3. 初期セットアップ – スレーブノードの追加(4/4)

Jenkins

- スレーブ（エージェント）追加完了！
  - 今回は、ソース管理サーバー（01 SCM）、ビルドサーバー（02 Build）、Libertyプロファイルサーバー実行環境（03 Production）の3ノードを追加しています。

S	名前 ↓	アーキテクチャ	クロック誤差	応答時間	空きスワップ容量	空きテンポラリ容量	空きディスク容量	
	01 SCM	Linux (amd64)	同期中	1089ms	3.87 GB	13.46 GB	13.46 GB	
	02 Build	Linux (amd64)	同期中	3083ms	3.87 GB	13.46 GB	13.46 GB	
	03 Production	Linux (amd64)	同期中	4047ms	3.87 GB	13.46 GB	13.46 GB	
	master	Windows 7 (amd64)	同期中	0ms	8.03 GB	3.22 GB	3.22 GB	
	データ取得	0.95 秒 前	50 ms 前	0.88 秒 前	0.84 秒 前	0.88 秒 前	0.88 秒 前	

3. 初期セットアップ – Maven連携設定

- ビルドサーバー上にインストールされているMavenの情報を登録します。

3. 初期セットアップ – 変数設定（オプション）

- ジョブ実行時に使用する変数の登録を行うことができます。
- 今回は、ジョブとして実行するシェル内で繰り返し使用する各種パス情報を変数として登録しています。

3. 初期セットアップ – その他（オプション）

- その他、以下のメニューからセキュリティ設定、ユーザー設定などを必要に応じて行ってください。

Arquillian

Arquillian概要（1/2）

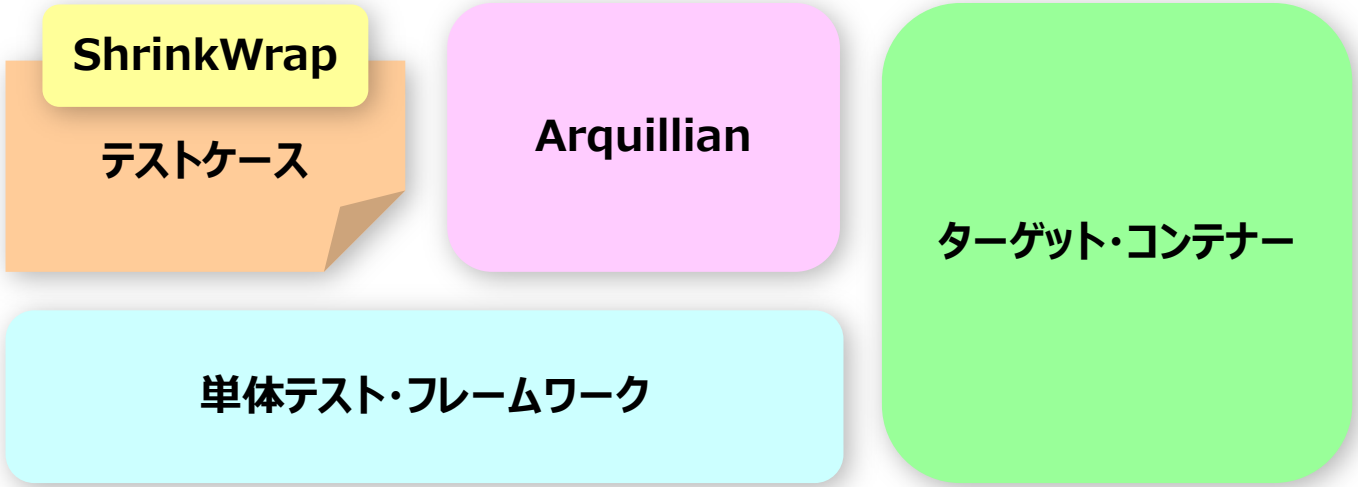
- Arquillianとは
  - Javaミドルウェア統合テストを簡素化することを目的にJBossコミュニティーで開発されているオープンソースのテストツール
  - 2012年4月にJBossコミュニティーが「Arquillian 1.0」をリリース
  - 主にJavaEEで利用することを目的としており様々なコンテナ環境上でのテストをサポート

Arquillian概要（2/2）

- Arquillianの機能
  - Arquillianを使用することでJavaEEアプリケーションサーバーのランタイムで動くテストを作成可能
  - Arquillianがテストに関連したコンテナライフサイクル管理や依存性クラスバンドルなどの機能を提供（開発者はテスト作成そのものにフォーカス）
  - アプリケーションサーバとの連携方法として下記3つをサポート
    - Remote
      - 既に起動しているアプリケーションサーバに対してデプロイとテスト実行を実施
    - Managed
      - Arquillianがアプリケーションサーバを起動してからテストを実行して最後にサーバをシャットダウン
    - Embedded
      - 組込みコンテナを使用して全てを同じJVMで実行

Arquillianのアーキテクチャー

- Arquillianは下記コンポーネントと連携してテスト環境を提供
  - 単体テストフレームワーク
    - ArquillianがテストコントローラーとなってJunit/TestNG等の単体テストフレームワークを利用
  - ShrinkWrap
    - テストの対象となるjar/war/ear等のアーカイブを作成するJavaのAPI
  - ターゲット・コンテナー
    - JavaEEアプリケーションサーバー/Servletコンテナー等のテスト実行環境
    - **ArquillianのコンテナーとしてLibertyプロファイルをサポート**



Arquillian使用手順 – 前提

- 一般的には以下のような流れでArquillianを使用

  Jenkinsから実行環境(例：開発環境のLibertyサーバー)を指定してテストを行うジョブを実行
  ↓
  実行環境(例：開発環境のLibertyサーバー)上でArquillianを使用してテスト実行

- この章では分かりやすいように以下のようにArquillianを使用する手順を紹介

  1つのマシン上に全てのコンポーネントを導入して
  Mavenでビルドしたアプリケーションのテストスイートに Arquillianを組み込んでテスト実行

①MavenでJavaプロジェクトを作成
②JavaプロジェクトをEclipseにインポート
③ArquillianのAPIを追加
④コンテナとしてLibertyサーバーを追加
初回セットアップ

⑤アプリケーションを開発
⑥Arquillianテストアプリケーションを開発
本番運用でサイクル実行

⑦テスト実行
⑧テスト実行結果を確認
Jenkinsからのジョブ実行

Arquillian使用手順 - ①MavenでJavaプロジェクトを作成

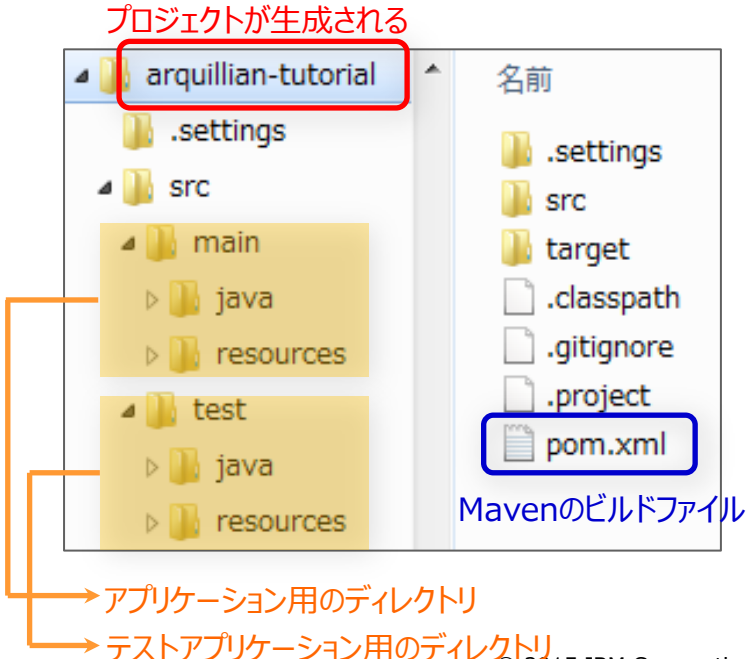
- Mavenをセットアップ
  - （Mavenの章を参照）
- MavenコマンドでArchetypeからJavaプロジェクトを作成

```
C:¥Tools¥apache-maven-3.3.3¥project>mvn archetype:generate -DarchetypeGroupId=net.avh4.mvn.archetype -
DarchetypeArtifactId=java-1.6-archetype -DgroupId=org.arquillian.example -DartifactId=arquillian-tutorial -
Dversion=1.0-SNAPSHOT
```

＜コマンド出力結果＞

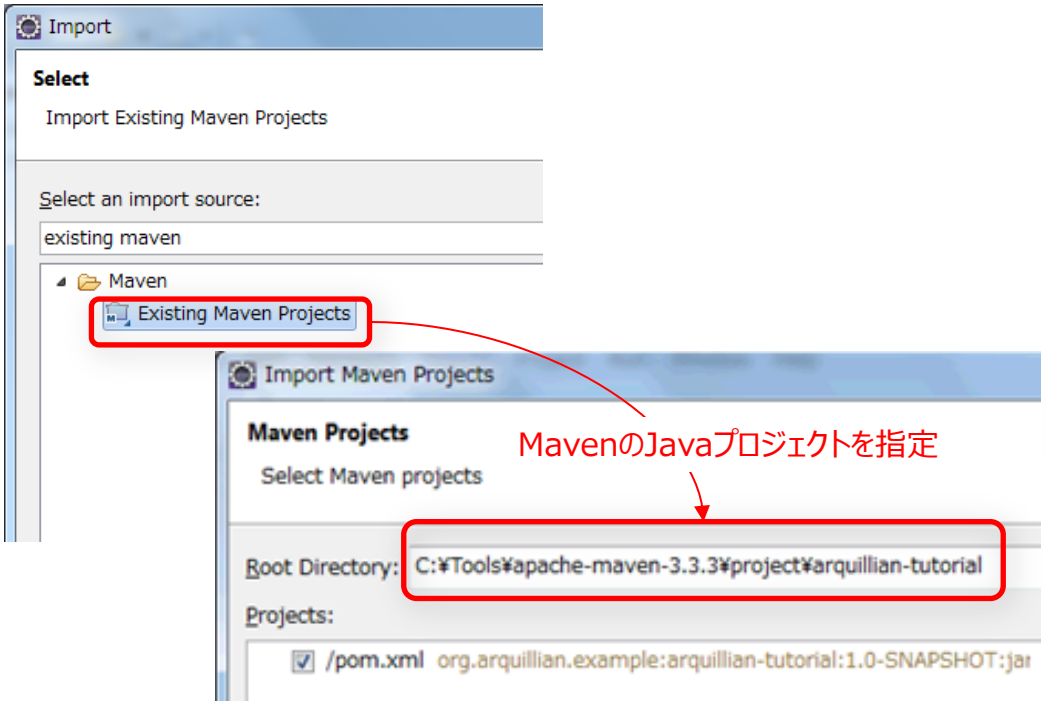
```
[INFO] Scanning for projects...
  ・・・・・＜略＞・・・・・
[INFO] ------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Archetype: java-1.6-
archetype:0.0.3
[INFO] ------------------------------------------------------------------------
[INFO] Parameter: groupId, Value: org.arquillian.example
  ・・・・・＜略＞・・・・・
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 14.611 s
[INFO] Finished at: 2015-10-07T15:04:36+09:00
[INFO] Final Memory: 49M/256M
[INFO] ------------------------------------------------------------------------
C:¥Tools¥apache-maven-3.3.3¥project>
```

＜生成されたディレクトリ構造＞

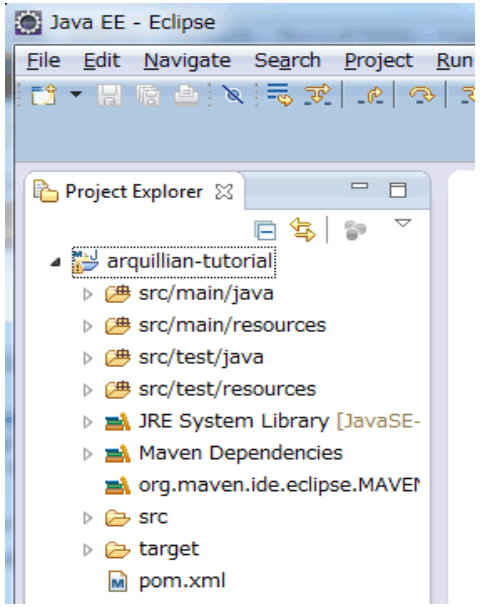


Arquillian使用手順 - ②JavaプロジェクトをEclipseにインポート

- Eclipseをセットアップ
  - Eclipseを導入
  - EclipseでMavenのJavaプロジェクトを扱うためにEclipseのMarketplace等から下記プラグインを導入
    - Maven Integration for Eclipse (m2e)
- Mavenで作成したJavaプロジェクトをEclipseにインポート
  - Eclipseのメニュー「File」>「Import」から「Existing Maven Projects」を選択してJavaプロジェクトを指定

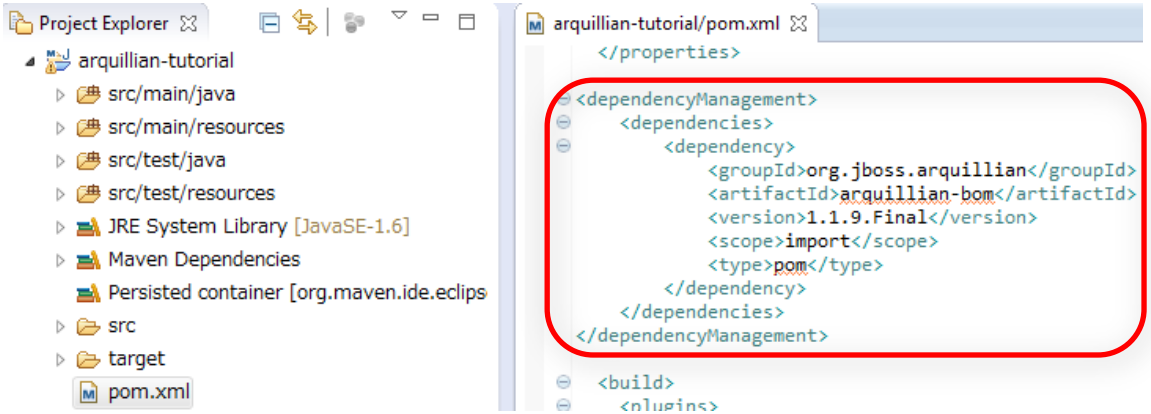


EclipseでMavenのJavaプロジェクトを参照できる

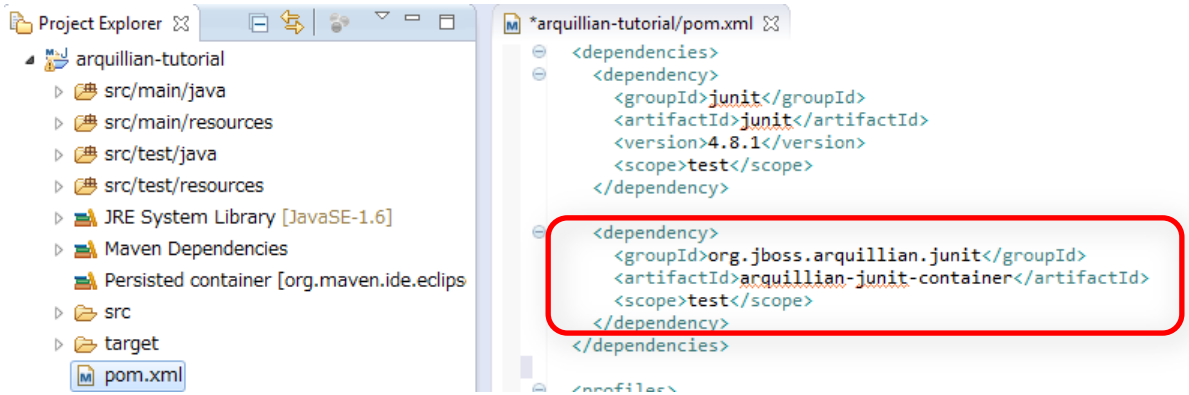


Arquillian使用手順 - ③ArquillianのAPIを追加

- Javaプロジェクト配下にあるpom.xmlを編集してMavenに下記アーティファクトを指定(Eclipse上でも編集可能)
  - ArquillianのBOM(Bill of Materials)　※<build>要素の上に追加

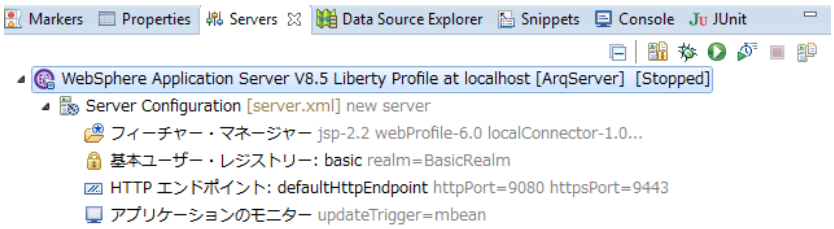


  - Arquillian-Junitインテグレーション　※<dependency>要素の下に追加



Arquillian使用手順 - ④コンテナとしてLibertyサーバーを追加

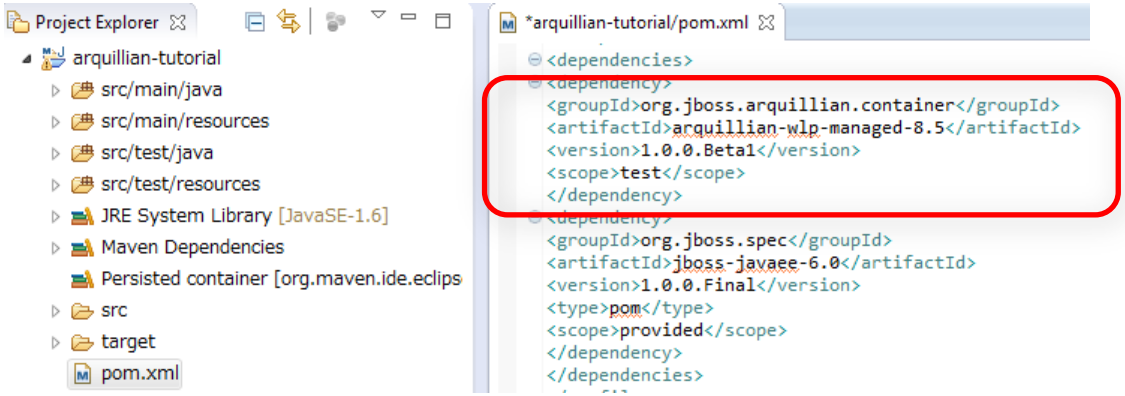
- Libertyサーバーを作成
  - Libertyサーバーを作成して下記機能を有効化
    - フィーチャー「localConnector-1.0」
      （アプリケーションに必要なフィーチャーも要追加）
    - applicationMonitorのMbeanを有効化



＜server.xmlの一部抜粋＞

```
<?xml version="1.0" encoding="UTF-8"?>
<server>
<featureManager>
<feature>jsp-2.2</feature>
<feature>localConnector-1.0</feature>
<feature>webProfile-6.0</feature>
</featureManager>
<applicationMonitor updateTrigger="mbean" />
</server>
```

- Javaプロジェクト配下にあるpom.xmlを編集してLibertyサーバーのコンテナアダプタを追加
  - arquillian-wlp-managed-8.5　※<dependency>要素の下に追加



Arquillian使用手順 - ⑤アプリケーションを開発

- テストの対象となるアプリケーションを開発
  ※ここでは例として「org.arquillian.example」パッケージに「Greeter」クラスを作成

単純に「Hello, xxxx!」というメッセージを返すアプリケーション

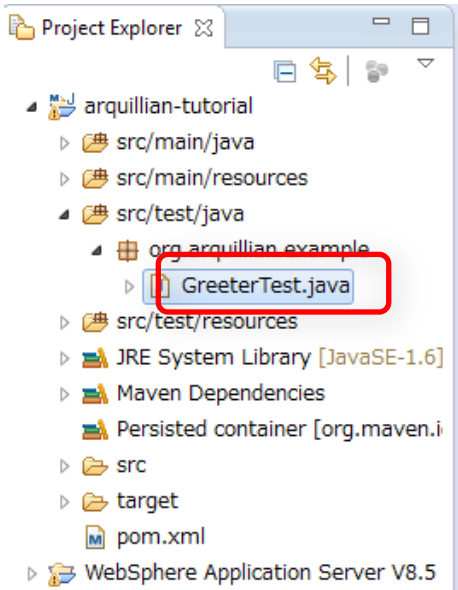
次に・・・・

このクラスがCDI(Contexts and Dependency Injection) beanとして注入された際に

正常稼動するかどうかを確認するためのテストアプリケーションを開発 → 次ページ

Arquillian使用手順 - ⑥Arquillianテストアプリケーションを開発

- アプリケーションをテストするためのテストアプリケーションを開発
  ※ここでは例として「org.arquillian.example」パッケージに「GreeterTest」クラスを作成



GreeterTest.java

```
package org.arquillian.example;

import org.jboss.arquillian.container.test.api.Deployment;
import org.jboss.arquillian.junit.Arquillian;
import org.jboss.shrinkwrap.api.ShrinkWrap;
import org.jboss.shrinkwrap.api.asset.EmptyAsset;
import org.jboss.shrinkwrap.api.spec.JavaArchive;
import org.junit.Assert;
import org.junit.Test;
import org.junit.runner.RunWith;
import javax.inject.Inject;

@RunWith(Arquillian.class)
public class GreeterTest {

    @Deployment
    public static JavaArchive createDeployment() {
        JavaArchive jar = ShrinkWrap.create(JavaArchive.class)
            .addClass(Greeter.class)
            .addAsManifestResource(EmptyAsset.INSTANCE, "beans.xml");
        System.out.println(jar.toString(true));
        return jar;
    }

    @Inject
    Greeter greeter;

    @Test
    public void should_create_greeting() {
        Assert.assertEquals("Hello, Earthling!",
            greeter.createGreeting("Earthling"));
        greeter.greet(System.out, "Earthling");
    }

}
```

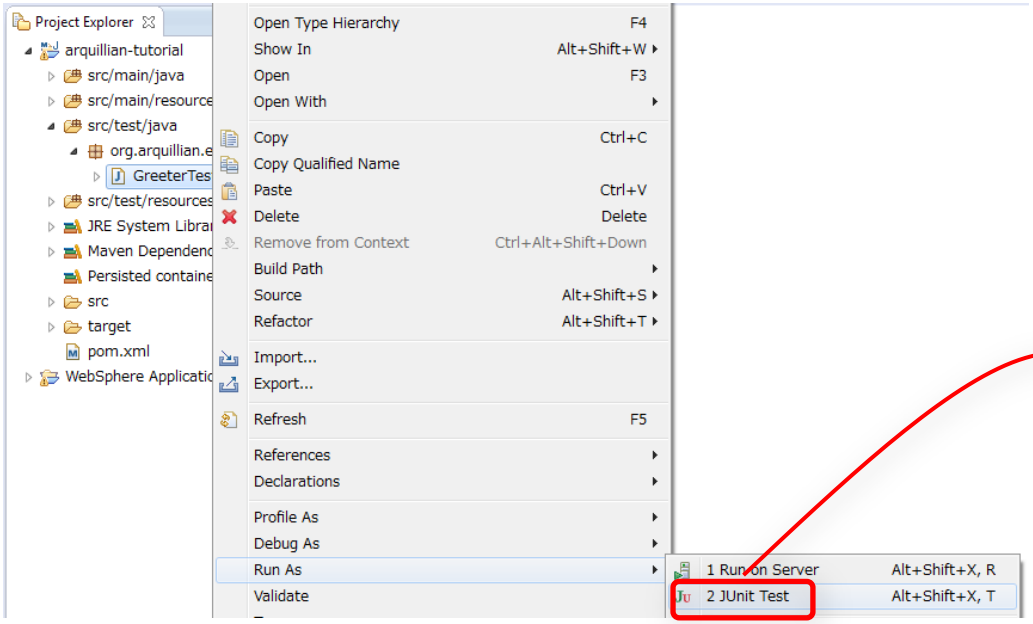
JunitにArquillianを使用することを伝える「@RunWith」

ShrinkWrapアーカイブを返すパブリックなスタティックメソッドの「@Deployment」

テストを行う「@Test」 ※ここではCDI bean として注入

Arquillian使用手順 - ⑦テスト実行

- 開発したArquillianテストアプリケーションを使用してテスト実行
  - Eclipseを利用する場合
    - Eclipse上でテストアプリケーションを選択して「Run As」>「Junit Test」を実行



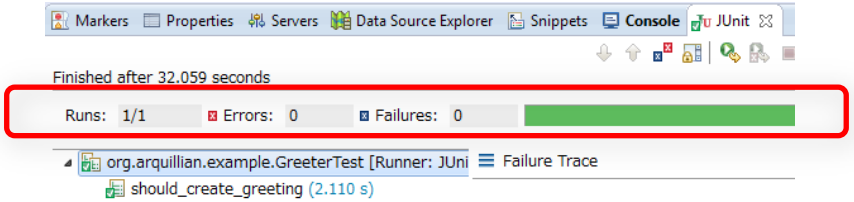
自動で下記処理が行われる
→Libertyサーバー起動
→アプリケーションがデプロイ
→テスト実行
→Libertyサーバー停止
→JUnitビューに結果表示

  - コマンドを利用する場合（Jenkinsから実行する場合など）
    - Mavenコマンドを実行

```
C:¥Tools¥apache-maven-3.3.3¥project>mvn test
```

Arquillian使用手順 - ⑧テスト実行結果を確認

- テスト実行結果を確認
  - Eclipseを利用する場合
    - Eclipse上のJUnitビューに表示される結果を確認



  - コマンドを利用する場合（Jenkinsから実行する場合など）
    - Mavenコマンドの実行結果出力メッセージを確認

```
C:¥Tools¥apache-maven-3.3.3¥project>mvn test
 [INFO] Scanning for projects...
  <途中省略>
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running org.arquillian.example.GreeterTest
  <途中省略>
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.626 sec
- in org.arquillian.example.GreeterTest

Results :
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

参考資料一覧

全般

- Libertyプロファイルとアジャイル開発/運用関連
  - Liberty and DevOps, continuous delivery and deployment environment
    - https://www.redbooks.ibm.com/redbooks.nsf/RedpieceAbstracts/sg248286.html?Open
  - Liberty in a DevOps Continuous Delivery Environment
    - http://www.redbooks.ibm.com/abstracts/redp5269.html?Open
  - Redbook "Configuring and Deploying Open Source with IBM WebSphere Application Server Liberty Profile"
    - http://www.redbooks.ibm.com/abstracts/sg248194.html?Open

ツール（Arquillian関連）

- Arquillian関連
  - IBM developerWorks「Tag Archives: Arquillian」
    - https://developer.ibm.com/wasdev/blog/tag/arquillian/
  - Arquillian Guides 「Getting Started」
    - http://arquillian.org/guides/getting_started_ja/
  - jboss.org「Arquillian: An integration testing framework for Java EE」
    - http://docs.jboss.org/arquillian/reference/1.0.0.Alpha1/en-US/html_single/

ツール（Maven関連）

- maven関連
  - MavenとLibertyプロファイルで実現するビルドプロセスの自動化
    - http://www.ibm.com/developerworks/jp/websphere/library/was/liberty_maven/
  - Installing Liberty with the Liberty Maven plug-in
    - https://developer.ibm.com/wasdev/docs/installing-liberty-liberty-maven-plug/
  - KnowledgeCenter: Using Maven to automate tasks for the Liberty profile
    - http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/ae/twlp_dev_maven.html

ツール（git関連）

- git関連
  - Git のコマンドだけでなく、その仕組みを学ぶ
    - https://www.ibm.com/developerworks/jp/devops/library/d-learn-workings-git/
  - GitHub wasdev
    - https://github.com/wasdev
  - Subversion ユーザーのための Git:
    - 第 1 回 Git 入門
      - http://www.ibm.com/developerworks/jp/linux/library/l-git-subversion-1/index.html
    - 第 2 回 細かく制御する
      - http://www.ibm.com/developerworks/jp/linux/library/l-git-subversion-2/index.html
  - IBM Open Source at GitHub
    - http://ibm.github.io/
  - サルでもわかるGit入門
    - http://www.backlog.jp/git-guide/
  - Git の基礎勉強 ~ Git によるバージョン管理 ~
    - http://tracpath.com/bootcamp/learning_git_firststep.html

ツール（Jenkins関連）

- Jenkins関連
  - IBM DevOps
    - https://www.youtube.com/channel/UCZjEZfm-WCJlQvWQ5y3nGkQ
  - DevOps with Liberty, Maven, and Chef
    - https://developer.ibm.com/wasdev/docs/devops-liberty-maven-chef-part-1/
  - Using Jenkins
    - https://wiki.jenkins-ci.org/display/JENKINS/Use+Jenkins

END