Version-updater Script Manual

If you have any questions about version-updater script please contact Irina Ivanova.

1. Usage manual

1.1 One module update ./update-version-tomcat.sh

1.1.1 Arguments

```
./update-version-tomcat.sh MODULE_NAME MODULE_VERSION JIRA_USERNAME [p]
```

Example: `./update-version-tomcat.sh admin 1.1.1.1 irina`

`MODULE_NAME` - required argument of eHealth module name, like admin or treatment; it's possible to configure autocomplete for this argument - see 3.1 Modules autocomplete configuration for details

`MODULE_VERSION` - required argument of module version, that user want to deploy

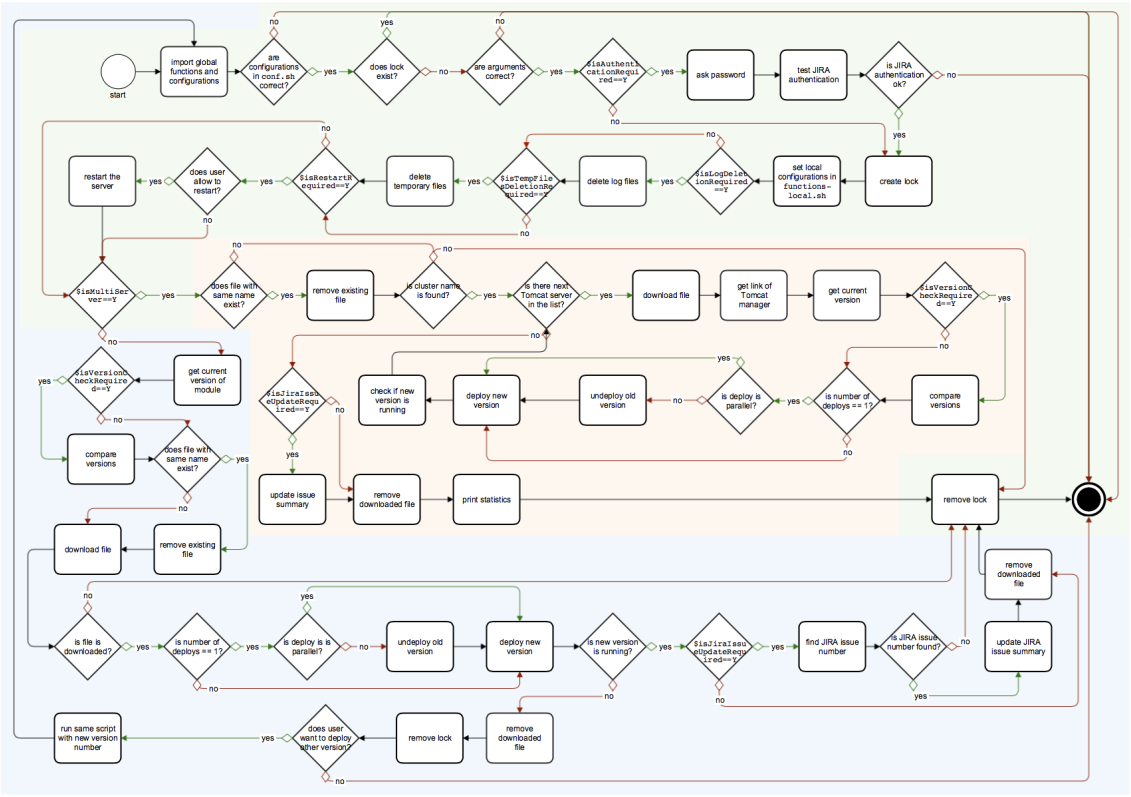
`JIRA_USERNAME` - required only if isAuthenticationRequired="Y" - see 2.2 Configurations in conf.sh for details

`[p]` - flag for parallel deployment - see 1.1.2 How it works for details

1.1.2 How it works

BPMN diagram for Tomcat

BPMN Diagram of One Module Update on Tomcat (`update-version-tomcat.sh`)



1.2 Update of several modules ./batch-update-versions-tomcat.sh

1.2.1 Arguments

```
./batch-update-versions-tomcat.sh JIRA_USERNAME [p]
```

Example: `./batch-update-versions-tomcat.sh irina`

`JIRA_USERNAME` - required only if isAuthenticationRequired="Y" - see 2.2 Configurations in conf.sh for details

`[p]` - flag for parallel deployment - see 1.2.2 How it works for details

Before running the script user has to describe modules and versions, that he want's to update, in the following file:

```
version-updater/batch-modules.txt
```

Content should be in the following format:

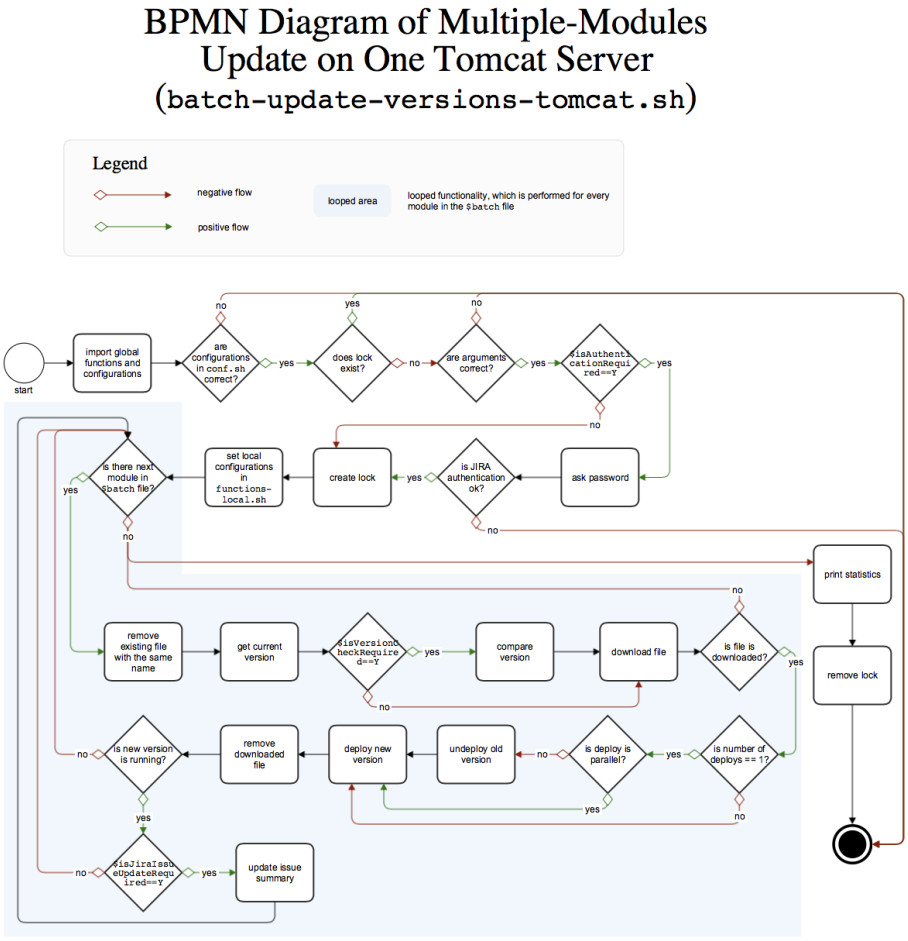
```
admin 1.1.1.1
billing 2.2.2.2
clinician-portal 3.3.3.3
tyk 4.4.4.4
```

NB! Order of modules is important - it defines in which order modules will be propagated. Right order can be found in version tracker: http://ehealthtest.webmedia.int:7070/module-tools/versiontracker

There is a script that generates `batch-modules.txt` file with all last stable versions - see 3.2 Getting stable versions with ./get-stable-versions.sh for details.

1.2.2 How it works

BPMN module for one Tomcat server



1.2.3 Statistics

At the end of all updates script shows statistics about which modules had errors and which modules were updates successfully.

Example:

```
************************************************
*******************STATISTICS*******************
CLUSTER ERRORS: 0
DOWNLOAD ERRORS: 1
         billing-2.2.2.2
UNDEPLOY WARNINGS: 0
DEPLOY ERRORS: 0
RUN ERRORS: 0
JIRA ERRORS: 0
VERSION WARNINGS: 0
DEPLOYED MODULES: 1
         admin-1.1.1.1
************************************************
```

Explanations:

`CLUSTER ERRORS` - displayed only if global variable $isMultiServer is Y (see 2.2 Configurations in conf.sh for details); list of modules where script couldn't find name of cluster where to deploy it

`DOWNLOAD ERRORS` - list of modules that script couldn't download

`UNDEPLOY WARNINGS` - displayed only if deploy is not parallel; list of modules that script didn't succeed to undeploy

`DEPLOY ERRORS` - list of modules that script didn't succeed to deploy

`RUN ERRORS` - list of modules that didn't run after deploy

`JIRA ERRORS` - displayed only if global variable $isJiraIssueUpdateRequired is Y (see 2.2 Configurations in conf.sh for details); list of modules where script didn't succeed to update JIRA issue summary

`VERSION WARNINGS` - displayed only if global variables $isVersionCheckRequired is Y (see 2.2 Configurations in conf.sh for details); list of modules, where deployed version cycle is different from old one

`DEPLOYED MODULES` - list of modules that were succesfully deployed

1.3 Lock

Both one-module and many-modules scripts create lock file at the beginning of work to avoid parallel activity on one server.

Name of lock files are defined in `update-version-tomcat.sh` and `batch-update-versions-tomcat.sh` scripts:

```
lock="UPDATING_$user-$module-$version.loc"
```

```
lock="UPDATING_BATCH_MODULES_$user.loc"
```

If this file exists it means that somebody is updating some module, so other user can't make updates at the same time. Script automatically removes lock file at the end of it's work.

NB! If script was terminated manually (for example Ctrl+C), then lock file should be also removed manually.

1.4 Logging

Script saves logs file that is defined in global variable `$log` - see 2.2 Configurations in conf.sh for details. By default log file is v`ersion-updater/update.log`

Log file contains following information:

- is it one module or many modules update
- timestamp
- username (if global variable `$isAuthenticationRequired` is `Y` - see 2.2 Configurations in conf.sh for details)
- is deploy is parallel or not
- status of downloading file
- status of deploying new version
- status of running new version
- status of updating JIRA issue summary (if global variable `$isJiraIssueUpdateRequired` is `Y` - see 2.2 Configurations in conf.sh for details)

You can color logs to simplify the reading - see Log Colorization With Rainbow for details.

1.5 Output colors

Explanations of output colors:

red - error, that can't be ignored (for example, file download is failed)

yellow - warning, that can be ignored (for example, old version undeploy failed - may be it didn't exist)

green - process is successfully completed (for example, file is downloaded)

cyan - info about performing some process (for example, script is downloading file right now)

gray background - breaklines

1.6 Notifications

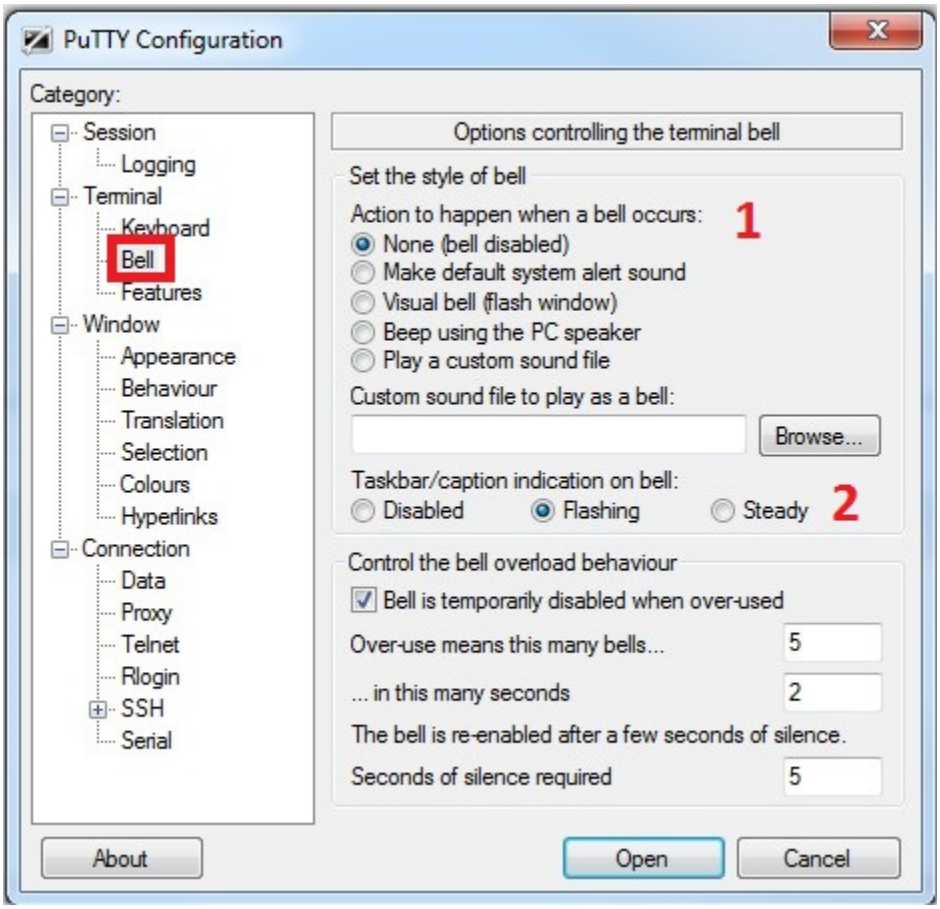
Script gives notifications in 2 cases:

- when it waits some response from user (JIRA password etc)
- when it finishes the work

The type of notification can be configured in terminal, that is used for updates.

1.6.1 Configurations in PuTTY

Configurations - Terminal - Bell



In first block you can choose audio effects and on the second block - visual effects.

NB! These configurations applies to the session, so if you change them you change all notifications in the session, not only version-updater's ones.

1.6.2 Configurations in Mac OS Terminal

Preferences - Profiles - Advanced - Bell

2 Installation and Configuration

2.1 How to install script on new environment

2.1.1 Easy download

There is a download directory in HG repo, where you can find zip files of stable version-updater versions.

NB! Zip files may not contain minor fixes, new versions will be generated mainly for new functionality! See download/changelog.txt for details

1. Download last zip file to the root directory
2. Unzip it
3. Modify following files to add environment-specific configurations (detailed explanations are inside the files):
    a. **`version-updater/conf.sh`**
    b. **`version-updater/functions-local.sh`**
    c. **`version-updater/jira-issues.txt`**
4. If files synchronization is needed, then add jobs to crontab - see 2.3 Synchronization for details

2.1.2 Old manual way

1. Create directory **`version-updater`** in the root
2. If you want to synchronize files configure cron jobs to automatically download them from repo - see 2.3 Synchronization for details
3. Download following files to **`version-updater`** directory from repo - these are files, that need to be downloaded only once and modified according to the environment:
    a. `batch-modules.txt`
    b. `jira-issues.txt`
    c. `jira-rest.txt`
    d. `conf.sh`
    e. `update.log`

f. `functions-local.sh`

```
wget http://ehealth.webmedia.ee/scripts/version-updater/batch-modules.txt
wget http://ehealth.webmedia.ee/scripts/version-updater/jira-issues.txt
wget http://ehealth.webmedia.ee/scripts/version-updater/jira-rest.txt
wget http://ehealth.webmedia.ee/scripts/version-updater/conf.sh
wget http://ehealth.webmedia.ee/scripts/version-updater/update.log
wget http://ehealth.webmedia.ee/scripts/version-updater/functions-local.sh
```

4. Modify file **`version-updater/jira-issues.txt`** and add module-issue mappings for all modules, that are deployed in envrionment
5. Configure global variables in **`version-updater/conf.sh`** - see 2.2 Configurations in conf.sh for details
6. Add content into file **`version-updater/functions-local.sh`** - you can see in other environments where script is already installed which content should it contain
7. After crontab job downloads synchronized files make script files .sh executable:

```
chmod +x *.sh version-updater/*.sh
```

In environment where Tomcat web manager is disabled (for example in production) you can use script for listing all deployed applications - see 3.3 Listing deployed applications with ./show-status-of-applications-tomcat.sh for details

NB! Before script first run be sure, that cron job downloads all files and all required files presents (especially **`extended-modules.txt`**)

2.2 Configurations in conf.sh

File **`version-updater/conf.sh`** contains global variables, that should be configured for every environment.

All comments and explanations are in the file itself.

2.3 Synchronization

Some common files (that all environments are using) can be synchronized, so if you need to modify or fix them you need to do it only once in repo. See Tortoise HG tutorial to learn how to work with HG repos.

You may not want to synchonize common files in production environments, because you need stable versions there and doesn't want to download every unchecked update that was made. In that case you need to manage common files manually there.

1. Synchronized files are located in HG repo `/EHEALTH/UTILS/scripts/version-updater` directory
2. At `ehealth proxy` server is described job, that every 10 minutes synchronize `UTILS` directory content, so fresh files (with 10 min delay) are available in the web http://ehealth.webmedia.ee/scripts/version-updater/
3. If envrionment should use synchronized files it should have cron jobs that download fresh files from the web
    a. In production servers jobs should be added under deploy user:

```
su - deploy
crontab -e
```

Example of crontab jobs. Files may vary in different environments - see 2.4 Files structure for details

```
crontab -e
```

```
*/10    *       *       *       *       cd ~/version-updater/; wget -N --no-cache http://ehealth.webmedia.ee/scripts/version-updater/functions-tomcat.sh; chmod +x version-updater/functions-tomcat.sh
*/10    *       *       *       *       cd ~/version-updater/; wget -N --no-cache http://ehealth.webmedia.ee/scripts/version-updater/readme.txt
*/10    *       *       *       *       cd ~/version-updater/; wget -N --no-cache http://ehealth.webmedia.ee/scripts/version-updater/functions.sh; chmod +x functions.sh
*/10    *       *       *       *       cd ~/version-updater/; wget -N --no-cache http://ehealth.webmedia.ee/scripts/version-updater/extended-modules.txt
*/10    *       *       *       *       cd ~; wget -N --no-cache http://ehealth.webmedia.ee/scripts/version-updater/update-version-tomcat.sh; chmod +x update-version-tomcat.sh
*/10    *       *       *       *       cd ~; wget -N --no-cache http://ehealth.webmedia.ee/scripts/version-updater/batch-update-versions-tomcat.sh; chmod +x batch-update-versions-tomcat.sh
```

2.4 Files structure

Files may depend on different environments.

Synchronized files - see 2.3 Synchronization for details

If you want to modify them you need to perform modifications in HG repo (not only in application server) or disable synchronization, otherwise changes will be overridden with file version in the repo.

- `version-updater/functions-tomcat.sh` - global functions that are specific for Tomcat 8
- `version-updater/functions.sh` - global functions, that are used in all servers
- `version-updater/extended-modules.txt` - file with list of modules, that have -lt extension. NB! It is crucial that content of this file is up-to-date. If some -lt module is migrated and this file is not updated, then script deploys wrong Estonian version
- `version-updater/readme.txt` - brief description of the script
- `update-version-tomcat.sh` - one-module updater
- `batch-update-versions-tomcat.sh` - batch modules updater

Local configuration files

Configuration files depend on specific environment. You can download their templates with detailed comments from HG repo (but you need to download them only once, no need in syncrhonization):

- `version-updater/conf.sh` - file with global variables that are specific for every environment
- `version-updater/functions-local.sh` - file with local functions that depend on users input (for example, to set module and version)
- `version-updater/batch-modules.txt` - file, where user should specify modules and versions that he wants to update with batch script (see 1.2.1 Arguments for details); file name and path can be changed with global variable `$batch`
- `version-updater/jira-issues.txt` - file, where are listed module and JIRA issue code mappings; file name and path can be changed with global variable `$issues`

Local output files

These files should be modified only by script, user shouldn't edit them.

- `version-updater/update.log` - file where script saves logs about all updates that were made with script; file name and path can be changed with global variable `$log`
- `version-updater/jira-rest.txt` - empty file, that script uses for performing REST request to JIRA; file name and path can be changed with global variable `$rest`

3 Additional extensions

3.1 Modules autocomplete configuration

It's possible to configure autocomplete for modules while running `./update-version-tomcat.sh` script. It's not version-updater functionality - it's server configuration.

On the server you need to add following code into **`~/.bash_profile`** file:

~/.bash_profile

```
# Autocomplete in version-updater
_complete_version_updater_modules() {
  COMPREPLY=()
  cur="${COMP_WORDS[COMP_CWORD]}"
  comp_modules=`cat ~/version-updater/autocomplete-modules.conf | \
            awk '{print $1}'
            `
  COMPREPLY=( $(compgen -W "${comp_modules}" -- $cur))
  return 0
}
complete -F _complete_version_updater_modules update-version-tomcat.sh
```

And you need to add **`version-updater/autocomplete-modules.conf`** file (example is available in HG autocomplete-modules.conf) with list of all modules, that you need to show in autocomplete.

3.2 Getting stable versions with ./get-stable-versions.sh

`get-stable-versions.sh` - script that generates `batch-modules.txt` file with all last stable versions. Script is located in HG and can be download from the web: http://ehealth.webmedia.ee/scripts/version-updater/get-stable-versions.sh

To use it, you need to define 2 variables: `$host` and `$env` of stable environment (see comments in the file).

It may be useful in production environment where usually are deployed versions, that are tested in test before.

3.3 Listing deployed applications with ./show-status-of-applications-tomcat.sh

There are scripts for listing deployed applications on the server (alternative for Tomcat web manager). Both are available in HG:

- `show-status-of-applications-tomcat.sh`
- `show-status-of-live-applications-tomcat.sh`

They are using version-updater's global variables and functions from `conf.sh` and `functions.sh`.

They can be useful if Tomcat web manager is disabled and there is no convinient way to list statuses od deployed applications.