Welcome on
developer.exoplatform.org!

Table of Contents

1. Sources Management 1
   1.1. Git - Repositories 1
      1.1.1. Platform codebase 1
   1.2. Git - Settings 2
      1.2.1. GitHub 2
      1.2.2. Git Configuration 3
      1.2.3. Git & IDEs 5
   1.3. Git - Workflow 6
      1.3.1. Repository architecture 6
      1.3.2. Contribution Workflow 7
      1.3.3. Branching model 8
      1.3.4. Feature Branch 8
      1.3.5. Fix Branch 9
      1.3.6. Stable Branch 11
      1.3.7. Integration Branch 13
      1.3.8. PoC Branch 13
      1.3.9. Release Process 14
      1.3.10. Community Contributions 14
      1.3.11. Improvement 14
   1.4. Git - Cheat Sheet 15
      1.4.1. Features 15
      1.4.2. Sources 21
2. IDE Settings to work on eXo Platform projects 21
   2.1. Eclipse - Settings 21
      2.1.1. Clean Up Settings: 21
      2.1.2. Code Templates Settings: 22
      2.1.3. Formatter Settings: 22
      2.1.4. Organize Imports Settings: 22
   2.2. IntelliJ - Settings 22
3. Build Management 22
   3.1. Maven - Setup guide 22
      3.1.1. Prerequisites 22
      3.1.2. Install Apache Maven 23
      3.1.3. Configure Apache Maven 23
4. Development 31
   4.1. Translations 31
      4.1.1. How to update an existing translation string? 31
      4.1.2. How to add a new translation string in an existing localization file? 32

4.1.3. How to delete a translation string? 32
4.1.4. How to add a new localization file? 32

This website is centralizing all resources useful for eXo developers and contributors.

 Download this documentation as PDF version

1. Sources Management

1.1. Git - Repositories

1.1.1. Platform codebase

1.1.1.1. Platform 5

eXo Platform is built from several projects stored in different **Git repositories**. You will find below the list of repositories used to build **eXo Platform 5**.

[Platform Components] | *plf5-components.png*

Figure 1. Overview of eXo Components for PLF 5.0

platform-public-distributions
: Main project used to package the standalone community edition based on Apache Tomcat 8.

platform
: eXo Platform

integration
: Integration of eXo services

calendar
: eXo Calendar

forum
: eXo Forum

wiki
: eXo Wiki

social
: eXo Social

ecms
: eXo Content Management System

commons
: Commons services

platform-ui
: eXo Platform UI customizations

GateIn Portal
: GateIn Portal

GateIn SSO
: GateIn SSO

GateIn PC
: GateIn PC

JCR
: eXo JCR

WS
: eXo WS

Core
: eXo Core

Kernel
: eXo Kernel

GateIn WCI
: GateIn WCI

1.1.1.2. Platform 4

[Platform 4.4 Components] | *plf4-components.png*

Figure 2. Overview of eXo Components for PLF 4.4

1.2. Git - Settings

1.2.1. GitHub

GitHub is the platform we chose to host our Git repositories. **To contribute to eXo projects you have to create a GitHub account.**

For eXo employees it is recommended to create one with the same username as the one provided by the company (GoogleApp Id) when you joined us. **You have to commit using your exoplatform.com email by setting it in your global git configuration or locally into each eXo repository clone** (see below).

To retrieve your rights to push changes or to pull private repositories you have to register your github account into your profile in our internal IDM My eXo.

GitHub provides a lot of documentation and especially an installation/configuration guide for each

operating system :

- Linux
- Windows
- MacOS

Please follow these guides to setup your environment.

1.2.2. Git Configuration

It is recommended to create an SSH private key (with a password) and to use it to access to your Git(Hub) repositories (see Github guides above).

By default you have at least to setup your default identity that will be used for all git instances on the system if you don't override them locally.

```
git config --global user.name "John Doe"
git config --global user.email "jdoe@exoplatform.com"
```

We recommend also :

- to configure git to activate use colors if you terminal supports it :

```
git config color.ui true
```

- to configure git to push only the current branch by default to avoid to send to the server some changes in others branches you weren't ready to share

```
git config --global push.default current
```

- to setup line endings preferences for Unix/Mac users

```
git config --global core.autocrlf input
git config --global core.safecrlf true
```

- to setup line endings preferences for Windows users

```
git config --global core.autocrlf true
git config --global core.safecrlf true
```

- to reduce the number of files to consider when performing rename detection during a merge. The merge is working pretty well on small repositories (with move and rename of files). But it's not working on large repositories as the detection of file renaming is $O(n^2)$, so we need to update some threshold (more explanations are available in this post :

http://blogs.atlassian.com/2011/10/confluence_git_rename_merge_oh_my/) :

```
git config --global merge.renameLimit 10000
```

- to configure git to add some command aliases to easily call them with `git <ALIAS_NAME>`. You just add a section `[alias]` in your `~/.gitconfig` file with entries like bellow

```
[alias]

[source]
##### Basic aliases
# Long status
st = status
# Short status
s = status -s
# Show all branches
br = branch -a
# Show branches with commit message
sb = show-branch
# Commit
ci = commit
# Checkout
co = checkout
# Show remote repositories
r  = remote -v
# Amend last commit
amend = ci --amend
# Removes files/directories from staging
unadd = rm -r --cached

##### Diff aliases
# Diff and show commands with word-diff style
wd = diff --word-diff
ws = show --word-diff
# Show diff before pull
do = diff ORIG_HEAD HEAD
# Show modified lines in the index
staged = diff --cached
# Show modified files
changes= diff --name-status -r
# Diff with statistics
ds = diff --stat -r

##### Log aliases
# Show HEAD commit
head = log --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr)
%C(bold blue)<%an>%Creset' --abbrev-commit --date=relative -n1
# Short one line logs with ref-names
l  = log --oneline --decorate=short
```

```
# Shows the last git logentry (hash, author, date commitmessage)
llm = log -1
# Short one line logs with ref-names and statistics
gl = log --oneline --decorate --stat --graph
# Short one line logs with ref-names (yellow, date (green) and author (blue)
glog = log --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s
%Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit --date=relative
# Show last commit
lc = log ORIG_HEAD.. --stat --no-merges
# Graph log with full commit message
glaaa = log --graph --abbrev-commit --date=relative

##### Misc
# Show last commiter
whois = !sh -c 'git log -i -1 --pretty=\"format:%an <%ae>\n\" --author=\"$1\"' -
# Show last commit message
whatis = show -s --pretty='tformat:%h (%s, %ad)' --date=short
# Hash of HEAD
h = rev-list --max-count=1 HEAD
# Show users which have commits in current branch
ul = !git log --format='%aN' | sort -u
# Number of commits in current branch
c  = !git log --oneline | wc -l
# Creates a tar.gz archive named after the last commits hash from HEAD! in the
directory above the repository
ahg = !git archive HEAD --format=tar | gzip > ../`git h`.tar.gz
# shows ignored directories
ignored = !git ls-files --others -i --exclude-standard --directory
# Move to the root of the repository
root = !cd $(git rev-parse --show-cdup)
# Show the root directory of the repository
sroot = rev-parse --show-toplevel
# Prune remote branches
prune-all = !git remote | xargs -n 1 git remote prune
# Show aliases
aliases = !git config --get-regexp 'alias.*' | colrm 1 6 | sed 's/[ ]/ = /'
# Show upstream for the current branch
upstream = !git for-each-ref --format='%(upstream:short)' `git symbolic-ref HEAD`
```

1.2.3. Git & IDEs

Git is natively supported by all IDE :

- Eclipse : EGit plugin bundled by default in the major part of eclipse distributions.
- IntelliJ : Native
- Netbeans : Native since 7.1

1.3. Git - Workflow

Our workflow is built on two principles :

- The **repository architecture** (development vs blessed) that has to be followed by **all projects involved in products** produced eXo (and thus that require to be supported).
- The **branching model** that has to be followed by *all projects* at eXo

1.3.1. Repository architecture

Aiming security and backup purpose, we are using two repository kinds (owned by two distinct github organizations).

These are `blessed` and `development` repositories hosted respectively on exoplatform and exodev organizations.

Development repositories are forked from blessed repositories.

This strategy is applied to all repositories/projects involved in eXo products. Others projects are using a single repository hosted on "exoplatform" organization.

[Git Organization] | *git-organization.png*

Figure 3. Git Repository Architecture

1.3.1.1. Development repository

This repository is the developers repository. The main development branch is `develop` branch. Depending the contribution kind (one shot contribution or feature contribution), the developer can use a dedicated `feature` branch if needed. The most of development activity will be done in this repository by a lot of contributors :

- `maintainers` (formerly known as `sl3`) : To develop fix. These developments are done in dedicated `fix` branches (more details below).
- `all development teams` : To do one shot contribution and handle feature branch life cycle.

In this repository, `develop` branch and `feature` branch will be under CI.

1.3.1.2. Blessed repository

In this repository you can find `stable` branches and release tags. Only some people are able to write on this repository :

- `maintainers` : To integrate pull request on stable branches.
- `release manager` : To perform release operations on stable (supported) branches.
- `keepers` : Repository keepers are `Project Leader`, `Team Leader` and `Technical Leader`. They are able to pull changes from dev to blessed when the develop is stable enough and they can process releases for non supported versions of products (alpha, beta, RCs, ...)

The continuous integration is applied on stables branches (`stable/1.0.x`, `stable/1.1.x`, etc) on

blessed repository.

1.3.2. Contribution Workflow

The contribution workflow mainly relies on Github Pull Requests. Here are the rules :

- `PR for everything`
  For every fix/feature, for all platform projects and supported addons, create a PR. Creating a PR is easy and allows to share, discuss on validate more easily the fix/feature. To create a PR :
  - create a branch locally with your fix and push it to Github
  - in Github, select your branch, and click on **New pull request** [Git PR Create]
  - select the right base branch (most of the time **develop**) and check that the PR contains only your commits [Git PR Create]
  - fill the description
  - click on **Create pull request**
- `PR on develop`
  PR must always be done on exodev:develop or exo-addons:develop, not on exoplatform:develop (we fix on develop first, then we backport to stable if needed). Remember that exoplatform:develop is read-only. There are only 2 exceptions :
  - the fix is different between develop dans stable - in such a case, 2 PRs are necessary, one on develop, one on stable
  - the bug only occurs on stable - in such a case, PR must be done on stable
- `Explain your PR`

  The description of the PR must always be filled to describe precisely what was the root cause of the problem and how it has been solved.
- `Update the PR, do not recreate it`

  To update a PR, just push a new commit in the same branch, no need to create a new branch and a new PR. Creating a new PR for each update prevents from following easily the discussion and the updates history.
- `PR must be validated by peers`

  When a PR is submitted, it has to be reviewed by at least one peer. When the PR is validated by the peer, the PR can be merged in the target branch.
- `Merge the PR correctly`

  Before merging the PR in the target branch, make sure the branch of the PR is up to date (push --force), otherwise the PR will not appear as **Merged** in Github.

  [Git PR Merged] | *git-pr-merged.png*
- `Clean your mess`

  Once the PR has been merged, delete the branch in Github, and close the PR if it is not already

marked as **Merged** or **Closed**.

`Code Review does NOT mean Test, Reviewers are NOT Testers`
The role of the reviewers is to review the code changes (code best practices, better/easier solution, ...). They do not necessarily have to test (they can if they want/need of course). The author of the PR must not rely on the reviewers to test it, he/she is responsible for that (and the QA people will help during their test campaigns).

1.3.3. Branching model

Branching model are 6 kinds of branch :

- `develop` : Develop branch contains the latest delivered development changes.
- `feature/xxx` : Feature branches are dedicated branch for one big feature (lot of commits), "xxx" is the feature name.
- `stable/xxx` : Stable branch are used to perform releases and write / accept fix. "xxx" is the stable version name (e.g 1.0.x).
- `fix/xxx` : Fix branch is dedicated to integrate bugfix on Develop branch. If needed the fix is then cherry pick on stable branch.
- `integration/xxx` : Integration branches are dedicated branch to automatic integration task (like Crowdin translation).
- `poc/xxx` : Poc branches are dedicated branch to develop a Prove of Concept (PoC).

1.3.3.1. Develop Branch

Develop branch contains the latest delivered development changes. This is our backbone where all the different fix and new feature are mixed with each other.

[Git Workflow - Develop Branch] | *git-workflow-develop-branch.png*

Figure 4. Git Workflow - Develop Branch

1.3.4. Feature Branch

Feature branches are dedicated branch to develop a new feature.

[Git Workflow - Feature Branch] | *git-workflow-feature-branch.png*

Figure 5. Git Workflow - Feature Branch

1.3.4.1. Actions

Create a new Feature Branch

When: A new feature is specified and planified.

Who: PL/TL

How:

- If you want the branch deploy on Acceptance, do not create the branch by yourself but create a SWF ticket on Jira for the full package (Branches+CI+Acceptance).
- If it's a local feature project without need for CI or Acceptance you can create it by yourself.

Rebase Develop to Feature Branch

When: Frequently

Who: Team responsible of the branch with support of team responsible each project.

How:

```
git checkout develop
git pull
git checkout feature/x
git rebase develop
git push --force
```

Merge Feature Branch to Develop

When: Feature has been successfully tested by FQA. VPs give a GO.

Who: Team responsible of the branch with support of team responsible of each project

How:

```
git checkout feature/x
git rebase -i origin/develop
(remove initial commit)
git checkout develop
git pull
git merge --no-ff feature/x
git push
```

Remove a Feature Branch

When: Just after the merge of the feature branch to Develop

Who: PL/TL

How: Create SWF ticket on Jira to remove the full package (Branches+CI+Acceptance).

1.3.5. Fix Branch

Fix Branch are dedicated branch to fix a bug. The validation process may be different if the bug has been raised by FQA/TQA or by SM.

A fix branch is always created from Develop branch (except exceptional circumstance: fix on stable only).

[Git Workflow - Fix Branch] | *git-workflow-fix-branch.png*

Figure 6. Git Workflow - Fix Branch

1.3.5.1. Actions

Create a Fix Branch

When: A Jira issue has been created, time to resolve it is already estimated.

Who: Team responsible to fix the issue.

How:

```
git checkout develop
git pull
git checkout -b fix/issue
git push
```

Merge a Fix Branch to Develop

When:

- If issue raised by TQA/FQA: After Engineering test
- If issue raised by SM: After SM test

Who:

- If issue raised by TQA/FQA: Team responsible to fix the issue
- If issue raised by SM: SM

How:

```
git checkout fix/issue
git pull
git rebase origin/develop
git checkout develop
git pull
git merge fix/issue --squash
git commit -a
git push
```

Remove a Fix Branch

When: After the merge of the fix branch to Develop

Who: Team responsible to fix the issue.

How:

```
git push origin --delete fix/issue
git branch -d fix/issue
```

1.3.6. Stable Branch

Stable branch are used to perform releases and write / accept fix.

[Git Workflow - Stable Branch] | *git-workflow-stable-branch.png*

Figure 7. Git Workflow - Stable Branch

1.3.6.1. Actions

Create a new Stable Branch

When: When create the first Release Candidate version

Who: SWF

How: With a script similar to [createFB.sh](github.com/exoplatform/swf-scripts/blob/master/createFB.sh)

Create a Fix Branch to fix Stable Branch

In exceptional circumstance

When: A fix need to be done on a specific version but not on the on development version (fix a performance issue for instance)

Who: Team responsible to fix the issue after a Go from SM.

How:

```
git checkout stable/4.1.x
git pull
git checkout -b fix/4.1.x-issue
```

Merge a Fix Branch to Stable

In exceptional circumstance

When: After SM test

Who: SM Team

How:

```
git checkout fix/4.1.x-issue
git checkout stable/4.1.x
git pull
git merge fix/4.1.x-issue --squash
git commit -a
git push
```

Remove a Fix Branch

When: After the merge of the fix branch to stable branch

Who: SM

How:

```
git push origin --delete fix/4.1.x-issue
git branch -d fix/4.1.x-issue
```

Perform a release

When: After FQA/TQA test campaign. VPs give a GO.

Who: Release managers

How:

```
git clone git@github.com:exoplatform/xxx.git
cd xxx
# You checkout the release branch on which you need to perform a release.
git checkout stable/A.B.x
# You follow the classical maven release process
mvn release:prepare
mvn release:perform
```

Move a release tag

In really special case (when the test campaign show a critical issue after tagging but before nexus publishing) release manager still can apply a last minute commit and move the tag.

When: After FQA/TQA test campaign. VPs give a GO.

Who: Release managers

How:

```
# After your commit, just delete the remote tag, and create another one in this way
git tag -d 1.0.0
git push origin :refs/tags/1.0.0
git tag 1.0.0
git push origin 1.0.0
```

1.3.7. Integration Branch

Integration branches are dedicated branch to automatic integration task (like Crowdin translation for instance).

1.3.7.1. Actions

Create a new Integration Branch

When: Whenever an integration is needed with a third-party system.

Who: SWF

How:

```
git checkout develop
git pull
git checkout -b integration/my-integration
git push
```

1.3.8. PoC Branch

`Engineering`

Poc branches are dedicated branch to develop a Prove of Concept (PoC).

[Git Workflow - POC Branch] | *git-workflow-poc-branch.png*

Figure 8. Git Workflow - POC Branch

1.3.8.1. Actions

Create a new PoC Branch

When: A new PoC is planified.

Who: PL/TL

How:

```
$ git checkout develop
$ git pull
$ git checkout -b poc/x
[Modify all pom: initial commit]
$ git add pom.xml
$ git commit -m "details"
$ git push
```

1.3.9. Release Process

A release must never involve a freeze of the develop branch. This section explain the release process to follow when doing an intermediate release (Milestone, Release Candidate) or the final release.

1.3.9.1. Intermediate Release

When: Product Leader give a go to do an intermediate release of PLF (Milestone, Release Candidate)

Who: PLF Team with support of team responsible of each project

[Intermediate Release] | *release-prepare-intermediate.png*

Figure 9. Intermediate Release

1.3.9.2. Final Release

When: Product Leader give a go to do the final release of PLF

Who: PLF Team with support of team responsible of each project

[Final Release process] | *release-prepare-final.png*

1.3.10. Community Contributions

Anyone with a Github account can contribute to eXo Platform. The only difference for people outside of the eXo Platform organization is they must sign a Contribution License Agreement. The Contributor License Agreement is needed to clarify the terms of usage of contributions by eXo Platform and the entire open source community.

The CLA must be printed, signed, then scanned as a PDF file and sent at cla@exoplatform.com.

1.3.11. Improvement

1.3.11.1. What is changing compare to 4.1

- Clean history by using git rebase.
- No more weekly merge between develop and master.
- All fixes are push firstly to develop branch. Then SM backport what they need to stable.

- Rebase develop to feature branch:
  - To do it regularly
  - To do it ONLY if develop branch is ok : build + acceptance are ok otherwise you'll distribute shitty code everywhere
  - To do it for all projects in a given FB at the same time (to keep the coherency)
- No more master branch on exodev repository. Master is only on blessed repository.

1.4. Git - Cheat Sheet

[Git Big Picture] | *git-overview.png*

1.4.1. Features

1.4.1.1. Create

From existing data

```
cd  <DIRECTORY>
git init
git add .
git commit
```

From existing repo

```
git clone <EXISTING_REPO> <NEW_REPO>
```

default protocol is ssh EXISTING_REPO is a local path or a remote URL, for examples :

- *path* : ~/repository.git
- *ssh* : git@host:repository.git
- *http(s)* : username@host/repository.git

1.4.1.2. Update

Fetch latest changes from origin

```
git fetch
```

this does not merge them

Pull latest changes from origin

```
git pull
```

does a fetch followed by a merge

In case of conflict, resolve the conflict and

```
git am ——resolve
```

1.4.1.3. Track Files

Start tracking files

```
git add <FILES>
```

Interactive selection of files to track

```
git add -i
```

Move/Rename a file

```
git mv <OLD_NAME> <NEW_NAME>
git mv <OLD_PATH> <NEW_PATH>
```

Stop tracking and delete files in working directories

```
git rm <FILES>
```

Stop tracking but keep files in working directory

```
git rm ——cached <FILES>
```

1.4.1.4. Commit

Commit all local changes

```
git commit -a +
```

Commit messages

Commits must relate to a JIRA issue. Convention for messages inspired by this article :

- The first line must be short (50 chars or less) and auto-descriptive in a format " ", for example `AM-101 : Fix the behavior of archives download`
- Write your commit message in the present tense: `Fix bug` and not `Fixed bug`.

- The second line is blank.
- Next lines optionally define a short summary of changes (wrap them to about 72 chars or so).

Example :

```
AM-101 : Fix the behavior of archives download

* —-no-cache has no effect on it
* The add-ons manager always retry to download the archive from the
downloadUrl. The download is skipped if the local file exists, it has
the same size as the remote one and its modifiedDate is > to the remote
one (It will allow to install new SNAPSHOTs without enforcing to
download the archive each time)
* The same behavior is applied for all URLs (http(s), file) +
```

1.4.1.5. Publish

Prepare a patch for other developers

```
git format-patch origin
```

Push changes to origin

```
git push origin <BRANCH>
```

Use option `--all` to push all local references (branches, tags..), `--tags` to push all tags, `--force` to push non fast-forward changes (must be avoided to not risk to loose commits)

Make a version or milestone

```
git tag <TAG_NAME>
```

1.4.1.6. Branch

List all branches

```
git branch
```

Switch to the BRANCH branch

```
git checkout <BRANCH>
```

Merge branch B1 into branch B2

```
git checkout <B2>
git merge <B1>
```

Create branch based on HEAD

```
git branch <BRANCH>
```

Create branch based on another

```
git branch <NEW> <BASE>
```

Delete a local branch

```
git branch -d <BRANCH>
```

Delete a remote branch

```
git push <origin> :<BRANCH>
```

1.4.1.7. Remote

List all your remote repositories

```
$ git remote -v
origin git@github.com:exodev/platform (fetch)
origin git@github.com:exodev/platform (push)
```

Add a new remote repository

```
git remote add upstream git@github.com:exoplatform/platform.git
```

Rename a remote repository

```
git remote rename upstream foo
```

Delete a remote repository

```
git remote rm upstream foo
```

1.4.1.8. Browse

Files changed in working directory

```
git status
```

Changes to tracked files

```
git diff
```

Changes between ID1 and ID2

```
git diff <ID1> <ID2>
```

History of changes

```
git log
```

Who changed what and when in a file

```
git blame <FILE>
```

A commit identified by ID

```
git show <ID>
```

A specific file from a specific ID

```
git diff <ID>:<FILE>
```

Search for patterns

```
git grep <PATTERN> <PATH>
```

1.4.1.9. Revert

Return to the last committed state

```
git checkout -f | git reset —hard
```

you cannot undo a hard reset

Revert the last commit

```
git revert HEAD
```

Creates a new commit

Revert specific commit

```
git revert <ID>
```

Creates a new commit

Fix the last commit

```
git commit -a ——amend
```

after editing the broken files

Checkout the ID version of a file

```
git checkout <ID> <FILE>
```

Restoring lost commits

So, you just did a `git reset --hard HEAD^` and threw out your last commit. Well, it turns out you really did need those changes. You'll never be able to implement that algorithm that perfectly twice, so you need it back. Don't fear, git should still have your commit. When you do a reset, the commit you threw out goes to a `dangling` state. It's still in git's datastore, waiting for the next garbage collection to clean it up. So unless you've ran a `git gc` since you tossed it, you should be in the clear to restore it.

```
git cherry-pick ORIG_HEAD
```

HEAD vs ORIG_HEAD

`ORIG_HEAD` is previous state of `HEAD`, set by commands that have possibly dangerous behavior, to be easy to revert them. It is less useful now that Git has reflog: `HEAD@{1}` is roughly equivalent to `ORIG_HEAD` (`HEAD@{1}` is always last value of `HEAD`, `ORIG_HEAD` is last value of `HEAD` before dangerous operation).

Removing a File from Every Commit (Powerful filter-branch)

This occurs fairly commonly. Someone accidentally commits a huge binary file with a thoughtless

`git add .`, and you want to remove it everywhere. Perhaps you accidentally committed a file that contained a password, and you want to make your project open source. filter-branch is the tool you probably want to use to scrub your entire history. To remove a file named passwords.txt from your entire history, you can use the `--tree-filter` option to `filter-branch`:

```
git filter-branch —tree-filter 'rm -f passwords.txt' HEAD
```

The `--tree-filter` option runs the specified command after each checkout of the project and then recommits the results. In this case, you remove a file called passwords.txt from every snapshot, whether it exists or not. If you want to remove all accidentally committed editor backup files, you can run something like `git filter-branch --tree-filter 'rm -f *~' HEAD`. Using `--index-filter` with `git rm` yields a significantly faster version. Like with using `rm filename`, `git rm --cached filename` will fail if the file is absent from the tree of a commit. If you want to `completely forget` a file, it does not matter when it entered history, so we also add `--ignore-unmatch`:

```
git filter-branch —index-filter 'git rm —cached —ignore-unmatch passwords.txt' HEAD
```

1.4.2. Sources

- Alex Zeitler Git cheat sheet
- Jan Krueger Git cheat sheet
- Git Ready
- Stackoverflow - HEAD and ORIG_HEAD in Git

2. IDE Settings to work on eXo Platform projects

2.1. Eclipse - Settings

Eclipse settings are available in /resources/ide/eclipse.

Configure Eclipse at "Window" → "Preferences..."

2.1.1. Clean Up Settings:

Java → Code Style → Clean Up

Choose: "Import..." exo-Java-CodeStyle-CleanUp.xml

[Eclipse Clean Up Settings] | *eclipse-Java-CodeStyle-CleanUp.png*

2.1.2. Code Templates Settings:

Java → Code Style → Code Templates

Choose: "Import..." exo-Java-CodeStyle-CodeTemplates.xml

[Eclipse Code Style Settings] | *eclipse-Java-CodeStyle-CodeTemplates.png*

2.1.3. Formatter Settings:

Java → Code Style → Formatter

Choose: "Import..." exo-Java-CodeStyle-Formatter.xml

[Eclipse Formatter Cleanup] | *eclipse-Java-CodeStyle-Formatter.png*

2.1.4. Organize Imports Settings:

Java → Code Style → Organize Imports

Choose: "Import..." exo-Java-CodeStyle-OrganizeImports.importorder

[Eclipse Organize Imports Settings] | *eclipse-Java-CodeStyle-OrganizeImports.png*

2.2. IntelliJ - Settings

Intellij IDEA users must install the plugin Eclipse Code Formatter and import Eclipse settings files from /resources/ide/eclipse.

- Create a specific profile (ex: "eXo")
- Use the file exo-Java-CodeStyle-Formatter.xml for "Eclipse Java Formatter config file"
- Use the file exo-Java-CodeStyle-OrganizeImports.importorder for "Optimize Imports from file"

[IntelliJ Eclipse Code Formatter] | *intellij-eclipse-code-formatter.png*

3. Build Management

3.1. Maven - Setup guide

3.1.1. Prerequisites

To build eXo projects you need to install a Java JDK 6, 7 or 8 depending of the Platform version you are targeting :

- **Platform 5.0**: `Java 8 / Maven 3.3.9`
- Platform 4.4+ : `Java 8 / Maven 3.2.x (PLF & GateIn components) / Maven 3.0.x (CF components)`
- Platform 4.1 / 4.2 / 4.3 : `Java 7 / Maven 3.2.x (PLF) / Maven 3.0.x (GateIn & CF components)`

- Platform 4.0.x : `Java 6`

3.1.2. Install Apache Maven

The version **3.3.9** of Apache Maven is currently recommended to build eXo projects.

The minimal version required is 3.0.4.
The version 3.2.2 must be avoided due to MNG-5663.

1. Download a fresh and clean copy of Apache Maven. It is critical to take a fresh copy to be sure that the file `apache-maven-X.Y.Z/conf/settings.xml` isn't modified !!
2. Unzip the distribution archive, i.e. apache-maven-X.Y.Z-bin.(zip|tar.gz) to the directory you wish to install Maven. The subdirectory `apache-maven-X.Y.Z` will be created from the archive. It is recommended to not store it under a path with spaces.
3. Add the path of `apache-maven-X.Y.Z/bin` in your `PATH` environment variable.
   - On Windows use the preferences screen of "Environment Variables"
   - On Linux/MacOS export the variable PATH from a startup script like .bash_profile (it depends of the OS and the shell you are using)
4. Add a system environment variable `MAVEN_OPTS` (it could be in a .profile startup script on Linux/MacOS operating systems or in the global environment variables panel on Windows)with the value `-Xshare:auto -Xms128M -Xmx1G -XX:MaxPermSize=256M`
5. Run `mvn --version` to verify that it is correctly installed.

Additional documentations

- Maven: The Complete Reference - Chapter 2. Installing Maven
- Maven website - Installing Maven

Maven and MacOSX

Apple provides for several years now, Maven as a standard tool in MacOSX distributions. You can see the version provided with `mvn --version`. You can find where the mvn script is located with which mvn (in theory in /usr/bin). It recommended to deactivate the one provided with MacOS X to use the one you'll define yourself. Take care if you upgrade your system, because Apple can restore the default version on your system. To deactivate the Maven version bundled in OSX, just remove the symlink : `sudo rm /usr/bin/mvn`

3.1.3. Configure Apache Maven

3.1.3.1. Basic setup

Since Platform 4, no specific settings are required to build eXo public projects. Just clone any project and launch `mvn install`

3.1.3.2. Advanced setup

If you need to release a project or to build a project relying on private or staging binaries you'll need to customize your maven settings.

Using our template create a file `settings.xml` in your `home directory` under a directory called `.m2` (if you already launched maven this directory must already exist and it should contain a subdirectory `repository` where Maven stores all artifacts it processes).

```
<?xml version="1.0" encoding="UTF-8"?>
<settings
  xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <servers>
    <!-- eXo Platform credentials -->
    <!-- Used to upload binaries on repository.exoplatform.org -->
    <server>
      <id>repository.exoplatform.org</id>
      <username><!-- Your eXo LDAP/Crowd Identifier --></username>
      <password><!-- Your eXo LDAP/Crowd Password --></password>
    </server>
    <!-- Used to download private binaries from repository.exoplatform.org -->
    <server>
      <id>exo.private</id>
      <username><!-- Your eXo LDAP/Crowd Identifier --></username>
      <password><!-- Your eXo LDAP/Crowd Password --></password>
    </server>
    <!-- Used to download staging binaries from repository.exoplatform.org -->
    <server>
      <id>exo.staging</id>
      <username><!-- Your eXo LDAP/Crowd Identifier --></username>
      <password><!-- Your eXo LDAP/Crowd Password --></password>
    </server>
    <!-- Used to download custom projects binaries from repository.exoplatform.org -->
    <server>
      <id>exo.cp</id>
      <username><!-- Your eXo LDAP/Crowd Identifier --></username>
      <password><!-- Your eXo LDAP/Crowd Password --></password>
    </server>
    <!-- Used to release projects on repository.jboss.org -->
     <server>
      <id>jboss-releases-repository</id>
      <username><!-- Your JBoss.org Identifier --></username>
      <password><!-- Your JBoss.org Password --></password>
    </server>
  </servers>
  <mirrors>
    <mirror>
      <id>exo-mirror</id>
```

```
      <mirrorOf>external:*,!exo.private,!exo.cp,!exo.staging</mirrorOf>
      <url>https://repository.exoplatform.org/public</url>
    </mirror>
  </mirrors>
  <profiles>
    <profile>
      <id>exo-central</id>
      <!-- This "hack" change the behavior of maven to let it use our public mirror
      as the central repository (with snapshots activation).
      The URL is never used and is overrided by the mirror entry.
      -->
      <repositories>
        <repository>
          <id>central</id>
          <url>http://fake</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>true</enabled>
          </snapshots>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>central</id>
          <url>http://fake</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>true</enabled>
          </snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>
    <!-- Specific settings used while releasing a project -->
    <profile>
      <id>exo-release</id>
      <properties>
        <gpg.keyname><!-- The GPG Key to use to sign eXo releases --></gpg.keyname>
        <gpg.passphrase><!-- The passphrase of your GPG Key --></gpg.passphrase>
      </properties>
    </profile>
    <profile>
      <id>exo-private</id>
      <!-- Repositories to download eXo private binaries -->
      <repositories>
        <repository>
          <id>exo.private</id>
          <url>https://repository.exoplatform.org/private</url>
```

```
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>true</enabled>
          </snapshots>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>exo.private</id>
          <url>https://repository.exoplatform.org/private</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>true</enabled>
          </snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>
    <profile>
      <!-- Repositories to download eXo custom projects binaries and products patchs
-->
      <id>exo-cp</id>
      <repositories>
        <repository>
          <id>exo.cp</id>
          <url>https://repository.exoplatform.org/cp</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>true</enabled>
          </snapshots>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>exo.cp</id>
          <url>https://repository.exoplatform.org/cp</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>true</enabled>
          </snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>
    <profile>
```

```
      <id>exo-staging</id>
      <!-- Repositories to download eXo staging binairies -->
      <!-- TAKE CARE TO ACTIVATE IT ONLY IF REQUIRED -->
      <!-- These repositories are delivering binaries marked as released but allowed
to be replaced -->
      <!-- Maven never updates released binaries thus you have to cleanup your local
repository to grab an updated version -->
      <repositories>
        <repository>
          <id>exo.staging</id>
          <url>https://repository.exoplatform.org/staging</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>true</enabled>
          </snapshots>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>exo.staging</id>
          <url>https://repository.exoplatform.org/staging</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>true</enabled>
          </snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>
    <profile>
      <id>jboss-staging</id>
      <!-- Repositories to download JBoss staging binairies -->
      <!-- TAKE CARE TO ACTIVATE IT ONLY IF REQUIRED -->
      <!-- These repositories are delivering binaries marked as released but allowed
to be replaced -->
      <!-- Maven never updates released binaries thus you have to cleanup your local
repository to grab an updated version -->
      <repositories>
        <repository>
          <id>jboss.staging</id>
          <url>https://repository.jboss.org/nexus/content/groups/staging/</url>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>jboss.staging</id>
          <url>https://repository.jboss.org/nexus/content/groups/staging/</url>
        </pluginRepository>
```

```
      </pluginRepositories>
    </profile>
    <!-- This profile is always activated and let you define properties for dependent
environment stuffs -->
    <profile>
      <id>local-properties</id>
      <properties>
        <!--

<exo.projects.directory.dependencies>${user.home}/Applications</exo.projects.directory
.dependencies>
        <exo.projects.app.tomcat.version>apache-tomcat-
6.0.29</exo.projects.app.tomcat.version>
        <exo.projects.app.jboss.version>jboss-
5.1.0.GA</exo.projects.app.jboss.version>
        -->
      </properties>
    </profile>
  </profiles>
  <activeProfiles>
    <!-- make these profiles active all the time -->
    <activeProfile>exo-central</activeProfile>
    <activeProfile>local-properties</activeProfile>
  </activeProfiles>
</settings>
```

In the `~/.m2/settings.xml` configuration file you have to fill your credentials to access to protected binaries or to publish artifacts on repository.exoplatform.org. For a better security, you can encrypt passwords in your settings file, however you must first configure a master password. For more information on both server passwords and the master password, see the Guide to password encryption and the dedicated chapter in the Maven Reference Guide. To release a project you have also to define the GPG key to use and its passphase (see the GPG setup guide).

When your setup is done you can activate the following profiles :

- `-Pexo-release` : Automatically activated while releasing projects. This profile is also activated on our CI server. Your GPG key must be configured (see the GPG setup guide).
- `-Pexo-private` : To access to private binaries on repository.exoplatform.org (eXo employees only).
- `-Pexo-staging` : To access to staging binaries (releases in validation) on repository.exoplatform.org (eXo employees only). **TAKE CARE TO ACTIVATE IT ONLY IF REQUIRED.** These repositories are delivering binaries considered by maven as released but allowed to be replaced. Maven never updates released binaries thus you have to cleanup your local repository to grab an updated version.
- `-Pexo-cp` : To access to custom projects binaries on repository.exoplatform.org (eXo employees only).
- `-Pjboss-staging` : To access to staging binaries on repository.jboss.org. **TAKE CARE TO ACTIVATE IT ONLY IF REQUIRED.** These repositories are delivering binaries considered by

maven as released but allowed to be replaced. Maven never updates released binaries thus you have to cleanup your local repository to grab an updated version.

3.1.3.3. Maven and GPG

Install gnupg for your OS.

MacOS

For homebrew users :

```
brew install gnupg
```

For macport users :

```
sudo port install gnupg
```

Others, you can use the native package.

Ubuntu

```
sudo aptitude install gnupg
```

Windows

TBD

Validate your installation

```
$ gpg --version
gpg (GnuPG) 1.4.10
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
http://gnu.org/licenses/gpl.html[http://gnu.org/licenses/gpl.html]
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Home: ~/.gnupg
Supported algorithms:
Pubkey: RSA, RSA-E, RSA-S, ELG-E, DSA
Cipher: 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH, CAMELLIA128,
 CAMELLIA192, CAMELLIA256
Hash: MD5, SHA1, RIPEMD160, SHA256, SHA384, SHA512, SHA224
Compression: Uncompressed, ZIP, ZLIB, BZIP2
```

Generate your private GPG Key

Configure GPG

Enforce the level of encryption. Edit `~/.gnupg/gpg.conf`

```
$ vi .gnupg/gpg.conf
```

At the end of the file add :

```
personal-digest-preferences SHA512
cert-digest-algo SHA512
default-preference-list SHA512 SHA384 SHA256 SHA224 AES256 AES192 AES CAST5 ZLIB BZIP2
ZIP Uncompressed
```

Generate the key

Launch the key generation

```
$ gpg --gen-key
```

ALWAYS SELECT DEFAULT CHOICES AND DON'T USE AN EMPTY PASSPHRASE

Enter your personal information like here :

- Real Name : Arnaud Héritier
- Comment : eXo Platform CODE SIGNING KEY
- Email Address : arnaud.heritier@exoplatform.com

Your key is created.

You can list the key you just generated with :

```
$ gpg --list-key
pub 4096R/2CF0CC82 2009-11-17
uid Arnaud Héritier (eXo Platform CODE SIGNING KEY) arnaud.heritier@exoplatform.com
sub 4096R/37540EAE 2009-11-17
```

You send your key to a PGP server (you use the ID from the "pub" line)

```
$ gpg --keyserver hkp://pgp.mit.edu[hkp://pgp.mit.edu] --send-keys 2CF0CC82
```

Your GPG key is now ready to be used

Configure your GPG Key for Maven

Fill the GPG keyname and passphrase in the exo-release profile of your maven settings like described in Advanced settings

```
 <profile>
   <id>exo-release</id>
   <properties>
     <gpg.keyname>2CF0CC82</gpg.keyname><!-- This is the public ID is displayed with
the gpg list-key command described above -->
     <gpg.passphrase>My awesome passphrase</gpg.passphrase>
   </properties>
 </profile>
```

Test it

Clone this project : `git@github.com:exodev/maven-sandbox-project.git`

Launch the command : `mvn install -Pexo-release`

You should see .asc files installed along others artifacts in the `target` directory of the project

To end tests, try to release this project : `mvn release:prepare` followed by `mvn release:perform`.

You are ready. Your environment is setup to do a release with GPG signature.

Don't forget to logon into https://repository.exoplatform.org and drop your staging repository

More info

- http://www.sonatype.com/people/2010/01/how-to-generate-pgp-signatures-with-maven/
- http://www.apache.org/dev/release-signing.html
- http://www.apache.org/dev/publishing-maven-artifacts.html#gpg
- http://maven.apache.org/plugins/maven-gpg-plugin

4. Development

4.1. Translations

All the translations are managed in Crowdin. For each PLF version, a Crowdin project is created and linked. A synchronization between the eXo source code and Crowdin is performed once per day.

4.1.1. How to update an existing translation string?

Updating a string must be done in Crowdin directly (if it is done in the source code, it will be reset by the next synchronization). Once validated the translated string will be pushed in the sources

during the next synchronization.

4.1.2. How to add a new translation string in an existing localization file?

In order to add a new string, simply add it in the english (en) localization file. It will automatically be added in Crowdin during the next synchronization. You can add the string in others localization files, for testing purpose, but it will be reset during the next synchronization.

4.1.3. How to delete a translation string?

Deleting a string must be done in the sources by simply removing the string in all the localization files containing this string. The string will be automatically deleted in Crowdin during the next synchronization.

4.1.4. How to add a new localization file?

All localization files of a given project must be referenced in the file translations.properties located at the root of the project. Each line must be formatted as follows:

```
crowdin-path=source-path
```

where

- **crowdin-path** is the path of the localization file in Crowdin relative to the root of the Crowdin project
- **source-path** is the path of the english (en) localization file in the project sources, relative to the value of the property "baseDir"

For example:

```
baseDir=platform/calendar/
webapp/CalendarPortlet.properties=calendar-
webapp/src/main/resources/locale/portlet/calendar/CalendarPortlet_en.properties
```