



**PHPCon**  
**Italia 2009**

## Advanced use of SVN in the life-cycle of web oriented projects

Emiliano `AlberT` Gabrielli  
*emiliano.gabrielli@deArchitettura.com*

CIO & CTO @  **Architettura.COM**<sup>®</sup>



## Intro::why another SVN-related talk

- underline, one time again, the necessity of adopting versioning tools in every project
- discuss some undervalued security-related points:
  - ACL granularity
  - roles separation in the development team
  - security in the data storage
  - avoiding osmotic migration of configuration vulnerabilities from the devel/test environment into production
- point out how to use SVN with an enterprise oriented mind
  - avoid OSS forking
  - increase SW lifetime and maintainability
  - automate daily quality related tasks

## Everybody needs a RCS

A Revision Control System makes the life easier in a number of situations:

- aims to standardize the work-flow of the developing team
- makes junior coders less fearful
- automates production of the project history documentation
- gives the team a powerful weapon against disasters
- aims to automate deploying, making releases, application versioning, etc
- makes it possible to have per-customer customized versions of the same project, without the headache of syncing updates, security patches and bug fix
- permits the inclusion of 3<sup>rd</sup> party OSS, without the needing of forking it
- makes our new features to OSS simply available for submission to the community

## This presentation is on SVN

Using SVN is a drug. Once you know it you can't do anything without it

```
$ mkdir IPC_09
$ cd IPC_09
$ svnadmin create SVN_REPO
$ mkdir presentation_init
$ cp ../PHPCon09_SVN_advanced.odp presentation_init
$ svn import ./presentation_init/ \
             file:///home/albert/Desktop/IPC_09/SVN_REPO
$ svn co file:///~/Desktop/IPC_09/SVN_REPO presentation
$ rm -rf presentation_init
```

## SVN::ACL granularity

Using SVN in combination with apache `mod_dav_svn` and `mod_authz_svn` makes it possible to have a fine-grain control on the access of every single member of the development team to the code repository

- don't give direct access (ssh, file-system, etc) to the repository
- implement apache-svn repository access
- configure users (username/password pairs)
- define groups of users based on their role
- deny access to every user/group on every repository
- grant a WL based access to each repository

## SVN::role separation - mod\_authz\_svn syntax

[groups]

<group\_name> = <user>[,<user>...]

...

[<repo\_name>:<path in repository>]

@<group> = [rw|r]

<user> = [rw|r]

\* = [rw|r]

\* → *Everyone*

r → GET, PROPFIND, REPORT, OPTIONS

w → MKCOL, DELETE, PUT, PROPATCH, CHECKOUT, MERGE, MKACTIVITY

**svn copy** and **svn move** require at least **w** on destination

## SVN::role separation - users, groups, repos and paths

- 2 types of rights
- 3 levels of access granting:
  - Users and groups
  - Repository
  - Path in repository
- We can define non-system users in a htpasswd-like way:
  - **htdigest [-c] passwordfile realm username**
  - **-c** flag creates a new file
- We can grant different access to different repositories
- For fine grain per directory r/w control we need mod\_authz\_svn

## SVN::security in the data storage

By defining **roles** and **ACL** we can reach a good isolation of security sensitive data in the repository tree:

- DB users/pwds
- Application level credentials (backend)
- Company accounts (SMS-GW uses/pwd)
- **Credentials on the production server !**

Creating a production branch we can easily achieve all the aboves by the matter of a simple svn merge:

- Deny access to */branches/prod* to everybody
- Grant *rw* permission to the production server *webmaster*
- Create */branches/prod* as a copy of */trunk* asap
- Granted people has to modify credentials only once!

## SVN::avoiding configuration vulnerabilities dev-to-prod migration

Creating a **production branch** we also grant against the evil of routine-bounded tasks typical of development environment:

- devel/test passwords are typically weak and .. shared
- It is possible to automate a forced check on commit time, to search for evil well known code
- A conscientious use of markers like “FIXME” may be a best practice
- Maybe useful in devel to intentionally have (and commit) some kind of tricky code, helper scripts, debugging and so on
- The aboves do not have to be in production !

## SVN::aim to be “enterprise”

By the effort of developing one time forever some helper script it is possible to dramatically simplify and automate the life-cycle of a project, as the life of the project manager and the team

- SVN helps to avoid errors in the development work-flow
- **Who helps us to avoid errors in the SVN management ?**
- **Branching** and **merging** can be error prone
- **Tagging** has to be a coherent process through the entire
- Import a **vendor drop** has to be safe and simple
- New people in the team have to be easily integrated and skilled
- New developers have to be productive not creating disasters

## SVN::aim to be “enterprise” (2)

- avoid OSS forking:
  - Branching and using the **vendor drop import** concept we can easily modify OSS to our needs, without having to fork the vendor
  - *svn\_load\_dirs.pl* is a script available on svn web site that helps in the vendor drop import task
  - Merging the vendor in our trunk we can get the new and maintain our customizations ... for free !
- increase SW lifetime and maintainability:
  - Forking makes impossible to re-sync with OSS updates
  - One day our fork will be much less smart than its OSS counterpart
  - That day we will have to make a choice: redo the job or freeze our features forever
- Using a few helper scripts we can automate daily tasks:
  - assure tags naming and comments are coherent
  - Deploy its a matter of execute an interactive script
  - Everything is proposed/defaulted and previewed

## deprj-tools::a set of script that makes the difference

- We have developed a set of well tested, solid and simply to use script that automates and makes interactive every step in the project life-cycle:
  - *deprj-svn\_vendor\_load.sh*: a wrapper over *svn\_load\_dirs.pl*, but it knows our repository structure and automates importing and merging
  - *deprj-svn\_prj\_tag.sh*: assure the current copy is a trunk checkout, tells the name of the last tag made, give the opportunity to list tags suggest 1\_0\_0 as default if no previous tag is present
  - *deprj-svn\_prod\_sync.sh*: makes production branch if it does not exists, makes the merge from the trunk at the revision of the last tag, marks the current sync revision, gives the comment for commit
  - *deprj-common.sh*: functions and variables sourced by every other script. This makes the code KISS and DRY



**PHPCon**  
**Italia 2009**

**Questions ?**

Emiliano `AlberT` Gabrielli  
*emiliano.gabrielli@deArchitettura.com*

CIO & CTO @  **Architettura.COM**<sup>®</sup>





**PHPCon**  
**Italia 2009**

## Advanced use of SVN in the life-cycle of web oriented projects

Emiliano `AlberT` Gabrielli  
*emiliano.gabrielli@deArchitettura.com*

CIO & CTO @  **Architettura.COM**<sup>®</sup>

