Structure of Projects in Version Control

Asked 16 years, 4 months ago Modified 11 years ago Viewed 13k times

20

I know there are at least 10 different ways to structure project in version control. I'm curious what some methods being used are and which ones work for you. I've worked with SVN, TFS, and currently/unfortunately VSS. I've seen version control implemented very poorly and just OK, but never great.

Just to get the ball rolling, here is a review of things I've seen.

This example is SVN-based, but applies to most VCS's (not so much to distributed version control).

1. branch the individual projects that are part of site /division/web/projectName/vb/src/[trunk|branches|tags]
2. branch the whole site, in the case I've seen, the whole site except for core components was branched. /division/[trunk|branches|tags]/web/projectName/vb/src/
3. Use main-line a default, only branch when necessary for **huge** changes.

svn version-control tfs branch project-structure

Share

Improve this question

Follow

edited Aug 21, 2013 at 9:20

oberlies

11.7k ● 5 ● 67 ● 115

asked Aug 19, 2008 at 20:01

oglester

6,655 ● 8 ● 44 ● 64

1 Please, add the label `"svn"`, confusing for git -users. – hhh Jul 7, 2012 at 2:32

@hhh I added a comment about distributed version control(DVC). Perhaps an answer about structur in DVC would be helpful to someone? – oglester Jul 18, 2012 at 16:39

9 Answers

Sorted by: Highest score (default)

10

We practice highly componentised development using Java, we have about 250 modules in trunk that have independent life cycles. Dependencies are managed through Maven (that's a best practice right there), every iteration (bi-weekly) actively developed modules get tagged with a new version. 3 digit version numbers with strict semantics (major.minor.build - major changes means backwards incompatible, minor changes mean backwards compatible and build number changes mean backwards and forwards compatible). Our ultimate

software product is an assembly that pulls in dozens of individual modules, again as Maven dependencies.

We branch modules/assemblies when we need to make a bug fix or enhancement for a released version and we can not deliver the HEAD version. Having tagged all versions makes this easy to do but branches still incur a significant administrative overhead (specifically keeping branches in sync with certain HEAD changesets) that are partly caused by our tools, Subversion is sub-optimal for managing branches.

We find that a fairly flat and above all **predictable** tree structure in the repository is crucial. It has allowed us to build release tools that take away a lot of the pain and danger from a manual release process (updated release notes, project compiles, unit tests run through, tag is made, no SNAPSHOT dependencies, etc). Avoid putting too much categorization or other logic in your tree structure.

We roughly do something like the following:

```
svnrepo/
  trunk/
    modules/
      m1/ --> will result in jar file
      m2/
      ...
    assemblies/
      a1/
      ...
  tags/
    modules/
      m1/
```

```
        1.0.0/
        1.0.1/
        1.1.0/
       m2/
      ...
    assemblies/
      a1/
        iteration-55/
        ...
branches/
  m1/
    1.0/
    ...
```

For external dependencies, I can not overemphasize something like Maven: manage your dependencies as references to versioned, uniquely identified binary artifacts in a repository.

For intenal module/project structure: stick to a standard. Uniformity is key. Again, Maven can help here since it dictates a structure. Many structures are fine, as long as you stick to them.

Share Improve this answer Follow

edited Jun 24, 2009 at 7:00

answered Aug 19, 2008 at 20:43

Boris Terzic

10.9k ● 8 ● 46 ● 60

Is there something like maven that even exists for .NET? I've been unable to unearth anything. – whaley Jun 15, 2009 at 14:11

NMaven specifically targets .NET (codeplex.com/nmaven), have not used it myself. At work we have .NET code that is built using normal Maven and some Visual Studio wrapping plugins. – Boris Terzic Jun 15, 2009 at 20:12

It looks like a good start that we're kicking off a new project with a structure similar to yours :) Out of curiosity, do you have a shared parent pom? If so, do you place the parent pom in the "modules" directory or as an actual directory inside "modules"? – aberrant80 Jun 23, 2009 at 8:25

We have a hierarchy of parent poms and we treat them just like we do modules: they each have their own "module" directory inside modules. Since Maven2 this is finally cleanly possible since the parent poms are inherited over the repository. In Maven1 you actually have to use relative paths and it becomes nasty. – Boris Terzic Jun 24, 2009 at 6:59

Thank you :D Just one more question if you don't mind. We've had to do some module-renaming right now (inappropriate initial names), and we have a little argument going on. If, say, your "trunk/modules/m1" needs to be renamed to "trunk/modules/m10", do you think "tags/modules/m1" should be renamed to "tags/modules/m10" or should "tags/modules/m1" be retained and a new "tags/modules/m10" be created? – aberrant80 Jun 24, 2009 at 10:37

7

Example for SVN:

trunk/

branch/

tags/

The trunk should be kept at a point where you can always push a release from it. There should be no huge gaping bugs that you know about(of course there will be eventually but that is what you should strive for).

Every time you need to make a new feature, do a design change, whatever, branch. Tag that branch at the start. Then when you are finished with the branch tag it at the end. This helps out with merging back into trunk.

Every time you need to push a release, tag. This way if something goes horribly wrong you can rollback to the previous release.

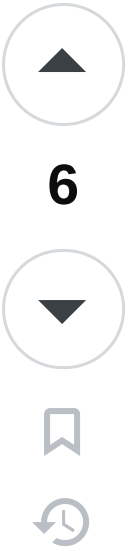
This setup keeps trunk as clean as possible and allows you to make quick bug fixes and push them out while keeping the majority of your development in branches.

Edit: For 3rd party stuff it depends. If I can avoid it I do not have it under source control. I keep it in a directory outside source control and include it from there. For things like jquery, I do leave it under source control. The reason is it simplifies my script for pushing. I can simply have it do an svn export and rsync.

edited Aug 19, 2008 at 20:30

answered Aug 19, 2008 at 20:09

mk.
26.2k ● 13 ● 39 ● 41



For my projects, I always use this structure.

> - trunk
>   - config
>   - docs
>   - sql
>   - initial
>   - updates
>   - src
>   - app
>   - test
>   - thirdparty
>   - lib
>   - tools
> - tags
> - branches

- config - Used to store my application config templates. During the build process, I take these templates and replace token placeholders with actual values depending on what configuration I am making the build.
- docs - Any application documentation gets placed in here.

- sql - I break my sql scripts into two directories. One for the initial database setup for when you are starting fresh and another place for my update scripts which get ran based on the version number of the database.
- src - The application source files. In here I break source files based on application and tests.
- thirdparty - This is where I put my third party libraries that I reference inside of my application and not available in the GAC. I split these up based on lib and tools. The lib directory holds the libraries that need to be included with the actual application. The tools directory holds the libraries that my application references, but are only used for running unit tests and compiling the application.

My solution file gets placed right under the trunk directory along with my build files.

Share Improve this answer

Follow

answered Aug 19, 2008 at 20:39

Dale Ragan

18.3k ● 3 ● 55 ● 71

how do you branch? if you branch just the src folder how do you handle your branch pointing to an older version of a thirdparty/lib? – wal Aug 28, 2011 at 13:30

I can appreciate the logic of not putting binaries in the repository but I think there is a huge advantage too. If you want to be able to pull a specific revision out from the

2

past (usually an older tag) I like being able to have everything I need come from the svn checkout. Of course this doesn't include Visual Studio or the .NET framework but having the right version of nant, nunit, log4net, etc. makes it really easy to go from checkout to build. This way getting started is as easy as "svn co project" followed by "nant build".

One thing we do is put ThirdParty binaries in a separate tree and use svn:external to bring it the version we need. To make life easy, we'll have a folder for each version that has been used. For example, we might bring in the ThirdParty/Castle/v1.0.3 folder to the current project. This way everything need to build/test the product is inside or below the project root. The tradeoff in disk space is well worth it in our experience.

Share Improve this answer

Follow

answered Aug 19, 2008 at 21:02

denis phillips

12.7k ● 5 ● 34 ● 47

2

As we have all the artifacts and construction in the same tree we have something like:

- Trunk
  - Planning&Tracking
  - Req
  - Design
  - Construction

    - Bin
    - Database
    - Lib
    - Source
- Deploy
- QA
- MA

Share Improve this answer

Follow

answered Sep 16, 2008 at 14:48

Mariano

3,008 ● 6 ● 26 ● 28

2 Why was this marked down? Seems like a decent structure even if it isn't one of the standard versions you see.
– Nathan Palmer Sep 5, 2009 at 18:55

2

I prefer fine-grained, very organized, self contained, structured repositories. There is a diagram illustrating general (ideal) approach of repository maintenance process. For example, my initial structure of repository (every project repository should have) is:

```
/project
    /trunk
    /tags
        /builds
            /PA
            /A
            /B
        /releases
```

```
            /AR
            /BR
            /RC
            /ST
    /branches
        /experimental
        /maintenance
            /versions
            /platforms
        /releases
```

`PA` means **pre-alpha** `A` means **alpha** `B` means **beta** `AR` means **alpha-release** `BR` means **beta-release** `RC` means **release candidate** `ST` means **stable**

There are differences between *builds* and *releases*.

- Tags under *builds* folder have version number corresponding to a pattern **`N.x.K`**, where `N` and `K` are integers. Examples: `1.x.0`, `5.x.1`, `10.x.33`
- Tags under *releases* folder have version number corresponding to a pattern **`N.M.K`**, where `N`, `M` and `K` are integers. Examples: `1.0.0`, `5.3.1`, `10.22.33`.

Recently I have developed training dedicated to Software Configuration Management where I describe version numbering approach and why exactly this repository structure is the best. Here are presentation slides.

There is also my answer on the question about 'Multiple SVN Repositories vs single company repository'. It might be helpful as long as you address this aspect of repository structuring in your question.

Share Improve this answer

Follow

edited May 23, 2017 at 11:45

Community Bot

1 ● 1

answered Jan 19, 2012 at 16:34

altern

5,949 ● 5 ● 46 ● 73

2 Could you please update the link to the diagram in first paragraph? – FvD Jun 22, 2013 at 13:29

1

I think the SCM policies and procedures a team adopts are going to be very dependent on the development process they are using. If you've got a team of 50 with several people working on major changes simultaneously and releases only occurring every 6 months, it makes a lot of sense for everyone to have his own branch where he can work in isolation and only merge in changes from other people when he wants them. On the other hand, if you're a team of 5 all sitting in the same room it makes sense to branch much less frequently.

Assuming you're working on a small team where communication and collaboration is good and releases are frequent, it makes very little sense to ever branch IMO. On one project we simply rolled the SVN revision number into the product version number for all our releases and we never even tagged. In the rare event that there was a critical bug found in prod we would simply branch straight from the revision that was released. But most of the time we simply fixed the bug in

the branch and released from trunk at the end of the week as scheduled. If your releases are frequent enough you'll almost never run into a bug that can't wait until the next official release.

I've worked on other projects where we never could have gotten away with that, but due to the lightweight development process and low ceremony we were able to use a lightweight version control policy very effectively.

I'll also mention that everything I've written is coming from an enterprise IT context where there's only a single production instance of a given code base. If I was working on a product that was deployed at 100 different customer sites the branching and tagging practices would have to be a little more strenuous in order to manage all of the independent update cycles across all the instances.

Share Improve this answer

Follow

answered Aug 19, 2008 at 20:24

Mike Deck

18.4k ● 16 ● 70 ● 94

1

> What about external dependencies such a the AJAXTookit or some other 3rd party extension that's used on several projects?

Source control is for source code, not binaries. Keep any 3rd party assemblies/jars in a separate repository. If you're working in the Java world try something like Maven

or Ivy. For .Net projects a simple shared drive can work well as long as you have decent policies around how it's structured and updated.

Share Improve this answer

Follow

answered Aug 19, 2008 at 20:29

Mike Deck

18.4k ● 16 ● 70 ● 94

0

We migrated from the bad world of VSS with one giant repository (over 4G) before switching to SVN. I really struggled with how to set up the new repository for our company. Our company is very "old" school. It's difficult to get change I'm one of the younger developers and I'm 45! I am part of a corporate development team that works on programs for a number of departments in our company. Anyway I set up our directories like this

```
+ devroot
    +--Dept1
        +--Dept1Proj1
        +--Dept2Proj2
    +--Dept2
        +--Dept2Proj1
    +--Tools
        +--Purchase3rdPartyTools
        +--NLog
        +--CustomBuiltLibrary
```

I wanted to include the ability to branch, but quite honestly that's just too much at this point. Couple things we still struggle with using this scheme.

- It's hard to fix production problems if you are working on a major product upgrade (ie because we don't do branching)
- It's hard to manage the concept of promoting from "Dev" to "Prod". (Don't even ask about promoting to QA)

Share Improve this answer

Follow

answered Aug 20, 2008 at 2:59

Monroecheeseman

2,021 ● 3 ● 18 ● 15