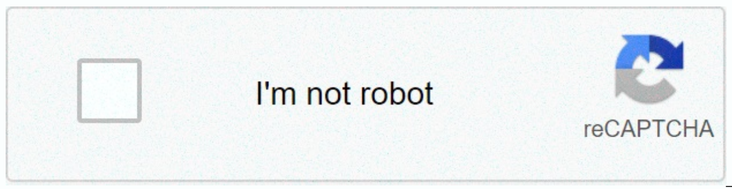

Continue

Tin electron configuration

A helpful reader provided a clean copy of one of the slides from the previous column: subversion's process area METALSMETHODS causes destruction (15-20 years) Idea 1. Religious politicization, commercialization, entertainment death wish 2. Educational tolerance, ignorance of relativity 3. Media monolith, manipulation, discredited and problem-free shortsighted eye 4. Culture fake heroes and role models addictive craze, mass structure 1. Law and order legislation, moral distrust justice 2 is not. Social relations rights vs. obligated individual responses 3. Security intelligence, police, military defense powerlessness 4. Internal Political Party, AnthagonisMudinitinitity 5. Foreign Salt.Friends Isolation Life 1. Family, society do not break loyalty (nation) 2. Health Sports, Medicare and Junk Food Emfiving Masse 3. Race Upper, Bible Slow?Genetics vs. Environmental Hatred, Division 4. Population - Land, Urbanization 5. Trade Unions vs. Destabilization of Social Damage (2-5 years) 1. Power struggle populism, irresponsible, power struggle Big Brother 2. The destruction of the economic negotiation process yields on Big Brother 3. Social fiber, low grassroots participation mobocracy 4. Foreign isolation, multilateral, and central communications. Prestige, a war-like siege crisis (2-6 months) normalization If you are the current release manager of the Subversion project or if you want, you should read and follow this procedure. $LastChangedDate: 2006-04-03 18:57:54 +0200 (Monday, April 03, 2006 The role of the release manager in the Subversion project is to handle the process of stabilizing, packaging, and releasing code to the public. If we were building an airplane, RM would be the one who would look at the construction checklist, paint the airline logo on the fuselage and deliver the finished unit to the customer. So there is no real development associated with being RM. All you need to do is uncoding being a public voice that coordinates people, concentrates information, and announces new stable releases. Many of the tasks that RM has to perform are repeatable and are not automated because users who have not yet disassembled and written the tool need to do the work or perform human validation to make automation a little extra. At this stage, you may consider RM's obligations unattractive. If you are looking for a position in a project that brings fame and fortune, it is better to implement what you really need to do in the trunk. If you're looking for something that really helps people who don't care about releases to focus on the code, RMing is for you. So, you are a release manager. What do you need to do? A new release branch is created for each new major and minor release. So, for example, a new releaseis created when preparing for the release of version 2.0.0 or version 1.3.0. However, when preparing for release 1.3.1 (patch version increments), the release branch created as of 1.3.0 is used. If you are preparing for a patch release, there is no release branch to create. All you have to do is select the location that you interrupted in the current minor version series release branch. The time it takes to create a new release branch is ambiguous at best. In general, there is a soft schedule of releasing a new minor version every six months. So, about 4 months after the previous minor release, it's a good time to start proposing a branch. However, remember that depending on the features under development, there is flexibility. If you agree that you need to create a new release branch, the release manager creates the following steps (replacing A.B with the version you are preparing, for example, 1.3, or 2.0, using .B . Edit subversion/include/svn_version.h on the trunk, where you increment the version number. Do not commit these changes yet. The trunk version number always reflects the major and minor versions that follow the version that created the branch (for example, 2.1.0 for the 2.0.x branch and 1.4.0 for the 1.3.x branch). Edit the changes for the trunk to introduce a new section of a future release. Beginning of section: Version A.B.0 (200X ?? ????? Leave the release date blank for released /.B/A .B.x). It will remain this until rolling time. Commit both of these changes in the following log message: Increment the trunk version number and introduce a new CHANGES section for future A.B.0 releases. * Sub-version/include/svn_version.h: Increase version number. * Changes: New .B in A.B.0. Once the release branch is created, no development occurs. Only changes made to various bookkeeping files, such as STATUS, are allowed, and changes merged from the trunk. The protocols used to accept or reject merging changes from trunks are of interest to all Subversion developers and are described in the release stabilization section of the Hacking Guide. The CHANGES file is a project change log file. Before release, you must list all changes since the most recent release. For patch releases, this is fairly simple: just go through the commit log of the branch after the last golden revision and note all the interesting merges. For minor and major releases, this is more complex: since the last release branch is forked, you need to traverse the trunk commit log and note all changes there. It's the same procedure, but much longer,It's more complicated because you have to exclude changesets that are returned to the previous release branch and released from there. Note that if necessary, the changes should always be edited in the trunk and merged into the release branch. It is very important to record all changes in all releases in a CHANGES file on the trunk for future reference so that the future release branch contains the sum of all previous change logs. The bulleted list for each change should be concise and no more than one line long. In some cases, it may be difficult, but it actually increases the overall readability of the document. Think to yourself: If the description takes more than one line, maybe I'm too detailed?svn log -rHEAD:BRANCH_POINT where BRANCH_POINT is a revnum where previous major/minor releases were branched from the trunk. This should give you all the changes made to the A.B.x line, including the backports made to the A.B.x branch. You should then delete the log of changes that have already been released in previous major/minor branch micro releases. Write a script that runs svn log --stop-on-copy on a branch of a previous release and parses revnum and removes it from the primary logging. (Carl and Ben used the emacs macro to do it, but I recommend writing a more general script.) Read the log from the oldest to the most recent log to summarize the points. The trick is to know the level of detail you write: you don't want to mention every little little commit, but you don't want to be too common too. Before you start a new section, read a few pages of the CHANGES file to set the filter level to ensure consistency. So the release branch is stabilizing and preparing to roll up the release. Before you can actually roll the archive, you need to set up a rolling environment for the white room. This environment should include not only pristine versions of some build tools, but also all related dependencies and other things bundled with the release tarball. You need to get the vanilla sauce tarball for the following build tool and then follow these steps: Autoconf 2.59 (2.60 is a significant compatibility break from the 2.5x series and is known to break the APR) Libtool 1.5.22 SWIG 1.3.25 It is important not to use the distribution shipping version of this software because it is patched in a non-portable way. The above version numbers typically need to be reviewed and increased in the latest stable upstream releases leading up to the A.B.0 release. A.B you want to change the version in the .x series, you should consider it carefully. It also gets the following source tarball, which is the part bundled with the Subversion release package:0.9.12) Latest Win32 Apache HTTP server release. From this zip file, use the APR, APR-util, and APR-iconv bundle libraries. You should use the win32 APR zip file, but be aware that there is currently an EOL issue with the win32 zip file. Check to see if this has been fixed and, if so, start using it. Once you have all of the latest Neon 0.25.x releases (currently 0.25.5) and the latest zlib releases (currently 1.2.3) (edit this document to reflect this), you can start the next step. Once that is done, you will have a build environment ready to roll the tar balls. Autoconf, Libtool, and SWIG: Select a directory that contains special build tools for Subversion RM obligations - for example /opt/svnrm. Configure, build, and install the three software --prefix=/opt/svnrm to verify that /opt/svnrm/bin is at the beginning of the PATH each time you run dist.sh. Prepare dependency sources: You must create two directories that hold dependencies that are repackad in Subversion. For example, you can select /opt/svnrm/unix dependencies and /opt/svnrm/win32-dependencies. Within Unix, unix APR and APR-util tarballs, neon and zlib tarballs should be unzipped to rename each top-level directory so that version numbers are not included. In Win32, you must unzip the Apache HTTP Server zip file, move the apr, apr-util, and apr-iconv directories from the srclib directory in the httpd zipfile to the dependent directory, and remove the remaining httpd extracts. Next, you need to copy the neon and zlib directories from the Unix dependency directory to the Win32 directory as well. .sh not support them, do not use symbolic links. Before rolling, first make sure that the latest version of the CHANGES file from the trunk is merged into the release branch and that the date at the top of the CHANGES matches the planned release date of the tarball. Build a tarball: From within the UNIX dependency directory, run the output of PATH=/opt/svnrm/bin:$PATH./dist.sh -v X.Y.Z -r 1234 -pr branch/X.Y.Z Watch dist.sh to make sure everything works smoothly. When it's done, you'll have a tarball in the cwd. Build a zip file: From within the Win32 dependent directory, run the output of PATH=/opt/svnrm/bin:$PATH./dist.sh -v X.Y.Z -r 1234 -pr branch/X.Y.Z -zip watch dist.sh to make sure everything goes smoothly. Once that is done, you will have a zip file in cwd. 9. Test one or both of the tarballs: a) tar zxvf subversion -X.Y.Z.tar.gz; for detailed instructions on configuring/building subversions, see cd subversion-X.Y.Z b) ./configure installation,Section III.B As mentioned above, if you install Apache in a location other than the default, the sameThe option used to set Apache. Also、-- enable-mod-activity automatically enables the required Subversion modules in the Apache configuration file. c) do) check e) (this activates mod_dav) f) check this, follow the instructions in the sub-version/test/cmdline/README, then launch Apache. If you want to maintain a development installation of apache, check the configuration file and make sure that you update to the new release area where you want to test tar-ball. (unless you match the name of the tree extracted from the tarball to httpd.conf, httpd.conf) g) g) you need to edit svncheck first, launch svnserve using these arguments Start svnserve: $ subversion/svnserve/svnserve -d -r \'pwd'/subversion/tests/cmdline -d tells svnserve to run as a daemon. After svnserve runs as daemon 'make svncheck', you need to run h) Then make sure that subversion/svn /svn co i) perl and python swig bindings are at least compiled in this environment. If this is not possible, ask another developer to review it. (See Bindings/Swiggs/Installations for more information) Make sure that the appropriate versions of Swigg, perl, and python have been detected. Next: Make Swig Pie Makeup Check Swig Pisudo Install Swig Pie Makeup Swig pl Make Check Swig pl sudo Install Swig pl j) Make sure the javahl binding is at least compiled. If this is not possible, ask another developer to review it. (See binding/java/javahl/README for more information) Make sure that the appropriate jdk has been detected and may re-run it with '--enable-javahl' and '--with-jdk='. Sign the release using GPG. gpg -b -- Armored Subversion -- X.Y.Z.tar.gz gpg -b -- ArmorEd Subversion -- X.Y.Z.tar.bz2 gpg -b -- Armor Capsizing -- X.Y.Z.zip Bless Release 11. Create tags using svn_version.h to reflect the final release. To do this, update the working copy to release revision 1234 (see the example below). Run svnversion to verify that there are no mixed working copies or that they do not contain modified work copies. Next, place the .h.svn_version dist file in the working copy and copy it from the working copy to the tag URL. For example, svn up -r 1234 svnversion .cp svn_version.h.dist subversion/include/svn_version.h svn cp .\ -m Tag release X.Y.Z and svn_version.h match tarball Note: Always create tags. 12. Upload a tar ball and . RM is given details on how to do this through a private channel. 13. Link to

TarballDownload page: a) Click on Download link (upper left frame)c) to add 'Source tarball' link (mainframe)d)" file Click on the link (top, 'file share') e) enter in the following fields: Name: subversion-X.Y.Z.tar.gz (replace X.Y.Z with release number) Status: Stable description: Subversion release X.Y.Z (MD5: <md5sum of= tarball=>) Content: (Select 'Link', enter f) Click Send 14. Bumps the svn_version.h of the original branch. Change the sub-version/svn_version.h. If you do 1.0.2, svn_version.h must have an appropriate value, such as 1.0.3. 15. Update the website. a) Properly edit and commit the www/project_status.html file in the release branch of /trunk *NOT*. Remember that you edited the search term at the end of the release issue link. If you used the 'svn switch' in the 3b above, you can use the svn switch to go back to /trunk and edit the www/project_status.html file properly. b) commit the best version www/project_packages.html) changes at the top of c). 16. Post the news item and send an announcement dev@ users@ announce@ list of news items. Be sure to include the URL and MD5 checksum in the announcement!note that the subversion.tigiris.org news item is in HTML format, not plain text. Tigis.org users with RM or higher status must approve the subversion.tigiris.org news item. For mail to the list, make sure that the mailer doesn't wrap more than 80 characters of line. You should also be notified of fresh meat for new releases: listed as a release manager or administrator for Freshfeat. Ask the user listed on the Subversion project page to access it. Your submission will also be tweaked by a crew of fresh meat before it is published. 17. Users with administrator access must upgrade svn.collab.net to head. (This is usually not a release manager.) 18. If you've made it this far, favorite_beverage your $100 million start now. Now. </md5sum>