ReleaseTodo

This page is to help a Lucene/Solr committer create a new release (you need committer rights for some of the steps to create an official release). It does not reflect official release policy - many of the items may be optional, or may be modified as necessary.

See the ASF Releases Policy page.

> ***NOTE: The new ReleaseWizard tool aims to automate and replace this document, please try it!***

Contents

Release Phases

The release process typically involves navigating these phases:

- **prerequisites**: making sure you have everything installed to manage a release
- **preparation**: work with the community to decide when the release will happen and what work must be completed before it can happen
- **branching and feature freeze**: creating a branch for this release
- **add new JIRA versions**: add version numbers to JIRA for the release after this one
- **run tests**: confirm that the tests pass within your release branch
- **build the release artifacts**: actually make the release tarballs and jar/pom files for Maven
- **test the release artifacts**: ensure that all tests pass when executed against your release candidate files
- **initiate a vote**: call a vote on the dev list to confirm the Lucene PMC is behind releasing these artifacts
- **publish artifacts to the ASF mirror system**: once a vote is completed, the artifacts can be pushed out to the world via the mirror system
- **update the website**: add references to the new release and publish the latest Javadocs
- **announce the release**: send emails announcing the release
- **post-release cleanup**: various tasks to remove old artifacts and update JIRA tickets, etc

Releases may be of three types:

- **major**: when the major version changes, e.g. 5.x -> 6.0.0 (includes JVM version changes, API breakages and major new features)
- **minor**: such as the transition from 5.3.x to 5.4.0 (includes API compatible updates)
- **bugfix**: such as 5.3.0 -> 5.3.1 (includes only bug fixes so can be a swap in replacement of its previous version)

The steps you will need to follow will, on certain occasions, differ depending upon which type of release you are undertaking.

Prior to every major or minor release (i.e. all except bugfix-only point releases) a feature freeze phase takes place for about 1-2 weeks. At the beginning of the feature freeze the development branch is copied to a release branch, and no commits are allowed to that release branch other than serious bug fixes, documentation or build updates.. This period of time should be used for extensive testing, documentation improvements and for cleaning up old JIRA issues.

Prerequisites

Python 3 (>= 3.4)

The Lucene deployment system makes extensive use of Python 3. You will need to have a copy installed locally to be able to complete any of the following steps. The minimum requirement is 3.4 as the release script uses the Python Enum package, which was only released in 3.4.

PGP/GPG Key

You will need to have a GPG key. You should:

- Make sure it is your 4K-bit key and not a 1K-bit key (esp. for ASF old timers). See http://www.apache.org/dev/release-signing.html for more information.
- Upload your key to the MIT key server, pgp.mit.edu.
- Make sure you put your GPG key's fingerprint in the **OpenPGP Public Key Primary Fingerprint** field in your profile on https://id.apache.org
- Optionally, you may want to edit your gnupg/gpg.conf file and add a default-key value with your Code Signing Key (as a HEX value). This will prevent you from having to specify -Dgpg.key=XXXXXX later when signing artifacts.

The tests will complain if your GPG key has not been signed by another Lucene committer - this makes you a part of the GPG "web of trust" (WoT). Ask a committer that you know personally to sign your key for you, providing them with the fingerprint for the key.

Preparation

On dev@lucene.apache.org , decide on:

1. which JIRA issues shall be committed before a release is made; set the appropriate "Fix Version" in JIRA for these issues
2. the date for branching from the unstable branch and the start of the feature freeze phase
3. the length of the feature freeze phase
4. a tentative release date

Branching & Feature Freeze

Branching

1. Run ant precommit to run a bunch of sanity & quality checks, then fix any problems that are found.
2. For the first release in a minor release series - i.e. X.Y.0 - create a minor release branch off the current major version branch, e.g. for minor release 5.5:
```
git clone https://git-wip-us.apache.org/repos/asf/lucene-solr.git lucene-solr
cd lucene-solr
git checkout branch_5x
git checkout -b branch_5_5
git push origin branch_5_5
```

Update Version Numbers in the Source Code

1. Add a new version for the next release using the addVersion.py script. If it is a bugfix release, we will be adding the bugfix version, otherwise we will add the version to come after the release we are producing.
    - If a bugfix release:

        - Do the following on the release branch only (note that later, as a post-release step, these will also be run on the stable and unstable branches; see the post-release section "Add the Released Version to the Stable and Unstable Branches" below):
            - python3 -u dev-tools/scripts/addVersion.py X.Y.Z
            - git add -u .; git commit; git push
        - Make sure that the backcompat index for the previous release has been added to the release branch. See the post-release section "Generate Backcompat Indexes" below - remember you'll be generating an index for the **previous** release.
    - If a minor release, you'll be adding the **next** minor version, not the version being released. Do the following on the stable branch and on the unstable branch:
        - python3 -u dev-tools/scripts/addVersion.py X.Y+1.0
        - git add -u .; git commit; git push
    - If a major release, you'll be adding the **next** major version, not the version being released. Do the following on the unstable branch:
        - python3 -u dev-tools/scripts/addVersion.py X+1.0.0
            - The script will print a list of items that need to be done manually for a major release bump.
        - git add -u .; git commit; git push
2. Sanity check the DOAP files under dev-tools/doap/: do they contain all releases less than the one in progress?

Jenkins Release Builds

After the branching is done, add Jenkins task for the release branch so that it runs the test, like other branches using the instructions on the JenkinsReleaseBuilds page.

Inform Devs of the Release Branch

1. Send a note to dev@ to inform the committers that the branch has been created and the feature freeze phase has started. Include Do's and Don'ts for the feature freeze phase:
    - No new features may be committed to the branch.
    - Documentation patches, build patches and serious bug fixes may be committed to the branch. However, you should submit **all** patches you want to commit to Jira first to give others the chance to review and possibly vote against the patch. Keep in mind that it is our main intention to keep the branch as stable as possible.
    - All patches that are intended for the branch should first be committed to the unstable branch, merged into the stable branch, and then into the current release branch.
    - Normal unstable and stable branch development may continue as usual. However, if you plan to commit a big change to the unstable branch while the branch feature freeze is in effect, think twice: can't the addition wait a couple more days? Merges of bug fixes into the branch may become more difficult.
    - **Only** Jira issues with Fix version "X.Y" and priority "Blocker" will delay a release candidate build.

Produce Release Notes

Get a draft of the release notes in place. These are typically edited on the Wiki.

1. Clone a page for a previous version as a starting point for your release notes. You will need two pages, one for Lucene and another for Solr. They will be on pages such as http://wiki.apache.org/lucene-java/ReleaseNoteXY(Z) and http://wiki.apache.org/solr/ReleaseNoteXY(Z)
2. Edit the contents of CHANGES.txt into a more concise format for public consumption
3. Ask on dev@ for input. Ideally the timing of this request mostly coincides with the release branch creation. It's a good idea to remind the devs of this later in the release too.

Add New JIRA Versions

1. Go to the JIRA "Manage Versions" Administration pages (https://issues.apache.org/jira/plugins/servlet/project-config/LUCENE/versions and https://issues.apache.org/jira/plugins/servlet/project-config/SOLR/versions) and add a new (unreleased) version for the next release on the unstable branch (for a major release) or the stable branch (for a minor release).

Run Tests

1. Check out the branch with: git clone https://git-wip-us.apache.org/repos/asf/lucene-solr.git lucene-solr; cd lucene-solr; git fetch upstream branch_5_5; git checkout branch_5_5
2. Build the code and javadocs, and run the unit tests: ant clean test, ant javadocs (in lucene/). Make sure that you are actually using the minimum compiler version supported for the release. For example, 5.x releases are on Java7 so make sure that you use Java7 for the release workflow.

3. Examine the results. Did it build without errors? Were there Javadoc warnings? Did the tests succeed? Does the demo application work correctly? Does Test2BTerms pass (this takes a lot of memory)?
4. Remove lucene/benchmark/{work,temp}/ if present

Building the Release Artifacts

If after the last day of the feature freeze phase no blocking issues are in JIRA with "Fix Version" X.Y then it's time to build the release artifacts.

1. It is recommended to clean your Ivy cache by executing rm -rf ~/.ivy2/cache before building the artifacts. This ensures that all Ivy dependencies are freshly downloaded, so we emulate a user that never used the Lucene build system before (this step ensures downloadability of all artifacts). If you have a ton of stuff in your ivy cache, it's a good idea to move the cache temporarily to another location and restore it after the RC has been built so you wouldn't have to download everything again. This expects you to use ant-1.8.x to build the artifacts. So, ensure that you have ant-1.8.x is installed and that is there in the path when you run the following commands.
2. Use dev-tools/scripts/buildAndPushRelease.py to build a release candidate and stage the results locally. Run buildAndPushRelease.py with `--help` to learn the available options. Here's an example of what was done for the 6.0.1 release:
    a. Build the release candidate: python3 -u dev-tools/scripts/buildAndPushRelease.py --push-local /tmp/releases/6.0.1 --rc-num 1 --sign <key-id>
    b. Run the smoke test script against the local release candidate: python3 -u dev-tools/scripts/smokeTestRelease.py /tmp/releases/6.0.1/lucene-solr-6.0.1-RC1-rev...
    c. Import the artifacts into SVN: svn -m "Lucene/Solr 6.0.1 RC1" import /tmp/releases/6.0.1/lucene-solr-6.0.1-RC1-rev... https://dist.apache.org/repos/dist/dev/lucene/lucene-solr-6.0.1-RC1-rev...
    d. If you have cancelled a prior release candidate, remove it from SVN: svn -m "Remove cancelled Lucene/Solr 6.0.1 RC1" rm https://dist.apache.org/repos/dist/dev/lucene/lucene-solr-6.0.1-RC1-rev...
    e. Don't delete these artifacts from your local workstation as you'll need to publish the maven subdirectories once the RC passes (see below).

Initiate a Vote

1. If the smoke test passes against the staged artifacts, send an email to the dev mailing list announcing the release candidate. You can use a subject of the form:
   `[VOTE] Release Lucene/Solr X.Y.Z RC?`

2. Here is a template you can use:
```
Please vote for release candidate ? for Lucene/Solr X.Y.Z

The artifacts can be downloaded from:
https://dist.apache.org/repos/dist/dev/lucene/lucene-solr-X.Y.Z-RC?-rev...

You can run the smoke tester directly with this command:

python3 -u dev-tools/scripts/smokeTestRelease.py \
https://dist.apache.org/repos/dist/dev/lucene/lucene-solr-X.Y.Z-RC?-rev...

Here's my +1
SUCCESS! [0:43:35.208102]
```

3. If the key you used to sign the release candidate artifacts has not been signed by other Apache committers, then testers may see the following warning:
```
verify trust
      GPG: gpg: WARNING: This key is not certified with a trusted signature!
```

Repeat if Needed

Should any issues be discovered, allow fixes to be committed to the release branch, and once complete, rerun tests, produce a new RC and call another vote, as described in the previous sections.

Publishing to the ASF Mirrors and to Maven Central

Once three PMC members have voted for a release, it may be published. You should wait at least 72 hours, not including weekend days. For instance, if you announce the RC on Friday, give until end of day Tuesday or Wednesday morning for the vote.

1. Announce that the vote has passed on the dev mailing list, ideally with subject beginning [RESULT]
2. Tag the release from the same revision from which the passing release candidate's was built:

```
git tag -a releases/lucene-solr/5.5.0 -m "Lucene/Solr 5.5.0 release"
2a228b3920a07f930f7afb6a42d0d20e184a943c

git push origin releases/lucene-solr/5.5.0
```

3. Delete the maven artifacts from the staging repo:
```
svn rm -m "delete the lucene maven artifacts" https://dist.apache.org/repos/dist/dev/lucene/lucene-solr-5.1.0-RC2-rev.../lucene/maven

svn rm -m "delete the solr maven artifacts" https://dist.apache.org/repos/dist/dev/lucene/lucene-solr-5.1.0-RC2-rev.../solr/maven
```
4. Move the new release artifacts to the release repo:
```
svn move -m "Move Lucene RC2 to release repo." https://dist.apache.org/repos/dist/dev/lucene/lucene-solr-5.1.0-RC2-rev.../lucene https://dist.apache.org/repos/dist/release/lucene/java/5.1.0

svn move -m "Move Solr RC2 to release repo." https://dist.apache.org/repos/dist/dev/lucene/lucene-solr-5.1.0-RC2-rev.../solr https://dist.apache.org/repos/dist/release/lucene/solr/5.1.0
```
5. Clean up the containing folder on the staging repo:
```
svn rm -m "Clean up the RC folder" https://dist.apache.org/repos/dist/dev/lucene/lucene-solr-5.1.0-RC2-rev...
```
6. Note at this point you will see the Jenkins job "Lucene-Solr-SmokeRelease-master" begin to fail, until you run the "Generate Backcompat Indexes" steps below!
7. Publish maven artifacts - in a source checkout do the following (note that this step will prompt you for your Apache LDAP credentials):
```
ant clean stage-maven-artifacts -Dmaven.dist.dir=/tmp/releases/6.0.1/lucene-solr-6.0.1-RC2-rev.../lucene/maven/ -Dm2.repository.id=apache.releases.https -Dm2.repository.url=https://repository.apache.org/service/local/staging/deploy/maven2

ant clean stage-maven-artifacts -Dmaven.dist.dir=/tmp/releases/6.0.1/lucene-solr-6.0.1-RC2-rev.../solr/maven/ -Dm2.repository.id=apache.releases.https -Dm2.repository.url=https://repository.apache.org/service/local/staging/deploy/maven2
```
8. Once you have transferred all maven artifacts to repository.apache.org, you will need to: log in there with your ASF credentials; locate the staging repository containing the release that you just uploaded; "close" the staging repository; wait and wait and keep clicking refresh until it allows you to: and then "release" the staging repository. This will cause them to sync to Maven Central. See here for more details.
9. Maven central should show the release after a short while, but you need to wait 24 hours to give the Apache mirrors a chance to copy the new release.
10. If you wish, use this script to continually check the number and percentage of mirrors (and Maven Central) that have the release: dev-tools/scripts/poll-mirrors.py -version 5.5.0.
11. Until all Maven mirrors have the release (which could take up to 24 hours), you should see the Lucene-Solr-SmokeRelease-master job fail. While waiting, this is a good time to execute the "Increment the version on the release branch" and "Generate Backcompat Indexes" steps as outlined below. (The latter step might not succeed immediately, due to the source tarball not available through the archive repository, but it will work after 1-2 hours.)

Update Website

Website += javadocs

The problem: Lucene's and Solr's voluminous per-release javadocs break the standard CMS process for the website (i.e., committer commits to the source tree; the CMS buildbot generates the site, then commits to the staging tree; committer reviews and then publishes to the production tree), because dynamic website updates, currently scheduled at 21 minutes after the hour on the hour, interrupt the extremely long commit times for javadocs being staged by buildbot, resulting in failed commits, caused by conflicts with the dynamic updates: by the time the buildbot-triggered commit has finished, its svn tree has been rendered stale.

The solution: skip committing javadocs to the source tree, then staging, then publishing, and instead commit javadocs directly to the production tree. Ordinarily this would be problematic, because the CMS wants to keep the production tree in sync with the staging tree, so anything it finds in the production tree that's not in the staging tree gets nuked. However, the CMS has a built-in mechanism to allow exceptions to the keep-production-in-sync-with-staging rule: extpaths.txt.

extpaths.txt lists paths in the production tree, relative to the project website's root directory, that are allowed to be out of sync with the staging tree.

For more info, see the following sections in the Apache CMS Reference:

- What is extpaths.txt?
- How do I publish generated docs (eg. doxygen)?
- A worked out example: How do I make the mail archives for my TLP accessible through the /mail URL?

Update extpaths.txt

1. svn co --depth=immediates https://svn.apache.org/repos/asf/lucene/cms/trunk/content website-source ('content' is a very generic name. So, we check it out to a self-explanatory name 'website-source')
2. cd website-source
3. Add Lucene javadocs dir: echo core/X_Y_Z >> extpaths.txt
4. Add Solr javadocs dir: echo solr/X_Y_Z >> extpaths.txt
5. Do a sanity check in the extpaths.txt to ensure that the above two lines were added at the end, and they are in their separate lines. This step is needed in case the file didn't have a newline character in the end to begin with, in which case just fix it by introducing a newline before core /X_Y_Z.
6. svn commit -m "Update CMS production sync exceptions for X_Y_Z javadocs" extpaths.txt
7. cd ..
8. rm -rf website-source

Push docs, changes and javadocs to the CMS production tree

1. Ensure your refridgerator has at least 2 beers - the svn import operation can take a while, depending on your upload bandwidth. (Data point: on a 5Mbps upload connection, it took me 11 minutes to upload the Lucene docs and 6 minutes to upload the Solr docs. - Steve)
2. Checkout the release tag created above and build the release documentation. Be sure to check that the version number is correct before pushing. Example:
```
git checkout releases/lucene-solr/X.Y.Z

ant documentation -Dversion=X.Y.Z

svn -m "Add docs, changes and javadocs for Lucene 6.0.1" import <checkoutroot>/lucene/build/docs
https://svn.apache.org/repos/infra/websites/production/lucene/content/core/6_0_1

svn -m "Add docs, changes and javadocs for Solr 6.0.1" import <checkoutroot>/solr/build/docs https://svn.
apache.org/repos/infra/websites/production/lucene/content/solr/6_0_1
```
3. Confirm you can browse to these URLs manually, and especially that solr javadocs link back to lucene's correctly. Examples:
    - http://lucene.apache.org/core/6_0_1
    - http://lucene.apache.org/solr/6_0_1

Update redirect to latest Javadoc

We make it possible to link to *latest* javadoc by providing redirect links for e.g. http://lucene.apache.org/solr/api/solr-core/ which will auto redirect to whatever is the latest released version, i.e. http://lucene.apache.org/solr/4_3_0/solr-core/ for 4.3.0. This is handled in .htaccess:

1. Goto CMS root http://lucene.apache.org/
2. Click on the CMS book marklet on your bookmarks bar. If you have not already installed the bookmark, go to CMS bookmarklet page and search for the link "ASF CMS" and follow the instructions.
3. Before changing anything here, click [Update this directory] to ensure you are operating on the latest version of the website.
4. Scroll down to the file .htaccess and click [Edit]
5. Locate the lines starting with RedirectMatch temp /core|solr/api/... and change the version url component to match X_Y_Z above. One of the RedirectMatch lines will be for the Solr reference guide, and will only have X_Y instead of X_Y_Z. That line should only be updated if a new reference guide is being released at the same time as this release.
6. Click Submit.
7. Click Commit, fill in a message and commit your change. No need to publish site yet - that will be done in next chapter.

NOTE As an alternative to CMS online editing, you can checkout the site from SVN and edit/commit .htaccess that way.

Update the rest of the website

NOTE: don't do this until you are ready to go to production. Every hour at :21 (e.g. 8:21, 9:21, ...) buildbot will commit all content from staging to production.

1. Once mirrors are ready, use the bookmarklet or whatever to do the minor changes and news blurbs and stuff.
    - The safest way is to svn checkout https://svn.apache.org/repos/asf/lucene/cms/trunk/ and use find/grep for the previous release, add entries, and then svn commit and look at the staging website (http://lucene.staging.apache.org). Javadocs won't work because they go directly to production (see above) but you can check everything else this way.
    - Be sure to update the version in content/latestversion.mdtext
2. Immediately after committing your changes, you should see the CMS buildbot kick off a new build here at https://ci.apache.org/builders/lucene-site-staging After that finishes, you should load http://lucene.staging.apache.org and verify your changes look OK, and iterate if not. Be sure your browser is not caching an old copy!
3. If for some reason your changes fail to show up at http://lucene.staging.apache.org, try editing a template file, e.g. add harmless whitespace to http://svn.apache.org/repos/asf/lucene/cms/trunk/templates/core-sidebar.html because this could provoke the buildbot into trying harder (do a "clean" build)
4. Once all is good on staging, publish the site, e.g. by visiting http://lucene.staging.apache.org, invoking the CMS bookmarklet, then clicking the publish link. Or just go straight here: https://cms.apache.org/lucene/publish

5. Wait for these changes to appear on both of Apache's main webservers (US: http://lucene.us.apache.org, EU: http://lucene.eu.apache.org, http://lucene.apache.org is dependent on your own geographic location, so the other mirror may still be outdated) before doing the next steps (see http://www.apache.org/dev/project-site.html for details on how the site is mirrored to Apache's main web servers). Once they appear, verify all links are correct in your changes!

As of the release of 7.0.1, the following files on the website must be modified (in addition to extpaths.txt which is already covered in previous steps) to include the new release version:

```
[cms-trunk] $ svn status
M       content/core/corenews.mdtext
M       content/core/documentation.mdtext
M       content/core/quickstart.mdtext
M       content/core/systemreqs.mdtext
M       content/latestversion.mdtext
M       content/mainnews.mdtext
M       content/solr/news.mdtext
M       content/solr/resources.mdtext
M       templates/_solr-downloads.mdtext
M       templates/core-sidebar.html
M       templates/corenav.mdtext
M       templates/mirrors-core-latest-redir.html
M       templates/mirrors-solr-latest-redir.html
M       templates/sidenav.mdtext
M       templates/solr-index.html
```

Update the project DOAP files

Update the Core & Solr DOAP RDF files on the unstable, stable and release branches to reflect the new versions (note that the website .htaccess file redirects from their canonical URLs to their locations in the Lucene/Solr Git source repository - see dev-tools/doap/README.txt for more info):

- dev-tools/doap/lucene.rdf
- dev-tools/doap/solr.rdf

Announce the Release

Release announcements can be shared/edited on the wiki at http://wiki.apache.org/lucene-java/ReleaseNoteXY(Z) and http://wiki.apache.org/solr/ReleaseNoteXY(Z)

For feature releases, your announcement should describe the main features included in the release; typically this is pulled from the wiki.

Mails to the announce@apache.org list must be sent from an @apache.org email address and should contain a signature.

Because you're likely not subscribed to the general, dev, and -user lists with your @apache.org address, sending the announcement email to those list with your @apache.org address will need to be moderated through, which will likely result in delayed transmission. To avoid this, send the emails to these lists using your subscribed email address, and then separately send the announcements to announce@apache.org .

Note: Copy-pasting from the release notes into Gmail might sometimes make Gmail (or any other client) send it as HTML formatted, which can break at the mailing list bot. Better to copy-paste the raw/edit text.

1. Announce the Lucene release on mailing lists general@lucene.apache.org , dev@lucene.apache.org , java-user@lucene.apache.org and announce@apache.org
2. Announce the Solr release on mailing lists general@lucene.apache.org , dev@lucene.apache.org , solr-user@lucene.apache.org and announce@apache.org
3. Add the new version to Wikipedia (english and maybe your own language)

Post-Release

Add the Released Version to the Stable and Unstable Branches

- **For bugfix releases only**: on both the stable and unstable branches:
    - python3 -u dev-tools/scripts/addVersion.py X.Y.Z
    - git add -u .; git commit; git push

Synchronize CHANGES.txt

Copy the CHANGES.txt section for this release back to the stable and unstable branches' CHANGES.txt files, removing any duplicate entries, but **only from sections for as-yet unreleased versions**; leave intact duplicate entries for already-released versions.

There is a script to generate a regex that will match JIRAs fixed in a release: releasedJirasRegex.py. The following examples will print regexes matching all JIRAs fixed in 5.5.2, which can then be used to find duplicates in unreleased version sections of the corresponding CHANGES.txt files, e.g. using a regex search in an IDE:

```
python3 -u -B dev-tools/scripts/releasedJirasRegex.py 6.6.0 lucene/CHANGES.txt

python3 -u -B dev-tools/scripts/releasedJirasRegex.py 6.6.0 solr/CHANGES.txt
```

Increment the version on the release branch

Add the next version after the just-released version on the release branch.

- python3 -u dev-tools/scripts/addVersion.py X.Y.Z+1
- git add -u .; git commit; git push

Generate Backcompat Indexes

After each version of Lucene is released, compressed CFS, non-CFS, and sorted indexes created with the newly released version are added to lucene /backwards-codecs/src/test/org/apache/lucene/index/, for use in testing backward index compatibility via org.apache.lucene.index. TestBackwardsCompatibility, which is also located under the backwards-codecs/ module. There are also three indexes created only with major Lucene versions: moreterms, empty, and dvupdates. These indexes are created via methods on TestBackwardsCompatibility itself - see comments in the source for more information.

There is a script (dev-tools/scripts/addBackcompatIndexes.py) that automates most of the process: it downloads the source for the specified release; generates indexes for the current release using TestBackwardsCompatibility; compresses the indexes and places them in the correct place in the source tree; modifies TestBackwardsCompatibility.java to include the generated indexes in the list of indexes to test; and then runs TestBackwardsCompatibility. Run this script on the stable branch, the unstable branch, and on the release branch. On each branch, after running the script, git add the generated indexes, then cd lucene/backwards-codecs and run ant test -Dtestcase=TestBackwardsCompatibility and make sure it passes before running git commit and git push.

TestBackwardsCompatibility will not test indexes for Version.LATEST, and will fail if it finds indexes from that version, so *before you run addBackcompatIndexes.py*, make sure dev-tools/scripts/addVersion.py has been run with the next version to be released on each of the branches - see item #3, about adding a new version, in the Branching & Feature Freeze section, above.

To print the script's usage, run it with the --help option: python3 -u dev-tools/scripts/addBackcompatIndexes.py --help

The script uses a scratch directory - /tmp/lucenebwc/ by default if you don't specify --temp-dir DIR - to store the source for the current release that it downloads, along with the generated indexes. This directory will be removed by the script unless you specify the --no-cleanup option, which is useful when running the script on multiple branches; in the example below, the --no-cleanup option is included on the non-final script invocations, but not on the final invocation, so that the scratch directory will be cleaned up after it's no longer needed.

Here's an example for the 7.0.0 release:

```
# If this were a bugfix release, e.g. 7.0.2, on the release branch we would first run addVersion.py
# if version 7.0.2 has not yet been added, and then addBackcompatIndexes.py for the previous release
# (7.0.1):
#
#    cd /path/to/lucene/working/tree
#    git checkout branch_7_0
#    git pull && git status   # Make sure there are no uncommitted changes
#    ant clean
#    python3 -u dev-tools/scripts/addVersion.py 7.0.2
#    git add -u .
#    git commit -m "Add version 7.0.2"
#    git push
#    python3 -u dev-tools/scripts/addBackcompatIndexes.py --no-cleanup 7.0.1
#    git add lucene/backward-codecs/src/test/org/apache/lucene/index/
#    git status # Make sure everything is ready to be committed
#    git commit -m "Add 7.0.1 back compat test indexes"
#    git push
#
# Then also execute all the commands below, substituting 7.0.1 where it says 7.0.0.

cd /path/to/lucene/working/tree
git checkout branch_7x
git pull && git status   # Make sure there are no uncommitted changes
ant clean
```

```
python3 -u dev-tools/scripts/addBackcompatIndexes.py --no-cleanup 7.0.0
git add lucene/backward-codecs/src/test/org/apache/lucene/index/
git status # Make sure everything is ready to be committed
git commit -m "Add 7.0.0 back compat test indexes"
git push

cd /path/to/lucene/working/tree
git checkout master
git pull && git status   # Make sure there are no uncommitted changes
ant clean
python3 -u dev-tools/scripts/addBackcompatIndexes.py 7.0.0
git add lucene/backward-codecs/src/test/org/apache/lucene/index
git status # Make sure everything is ready to be committed
git commit -m "Add 7.0.0 back compat test indexes"
git push
```

When doing a major version release, eg. 8.0.0, you might also need to reenable some backward compatibility tests for corner cases. To find them, run grep -r assume lucene/backward-codecs/, which should find tests that have been disabled on master because there was no released Lucene version to test against.

Update JIRA Versions

1. Go to the JIRA "Manage Versions" Administration pages (https://issues.apache.org/jira/plugins/servlet/project-config/LUCENE/versions and https://issues.apache.org/jira/plugins/servlet/project-config/SOLR/versions). Next to the version you'll release, click the gear pop-up menu icon and choose "Release". It will ask you for the release date -- enter it. It will give the option of transitioning issues marked fix-for the released version to the next version, but do **not** do this as it will send an email for each issue -- we'll address that separately.
2. Go to JIRA search in both Solr and Lucene and find all issues that were fixed in the release you just made, whose Status is Resolved. This URL may work (but edit the fixVersion part): (https://issues.apache.org/jira/issues/?jql=project+in+(LUCENE,SOLR)+AND+status=Resolved+AND+fixVersion=6.0.1). Do a bulk change (Under Tools... menu) to close all of these issues (this is a workflow transition task). Uncheck the box that says "send an email for these changes".
3. Do another JIRA search to find all issues with Unresolved Resolution and fixVersion of the release you just made. Note that Jira can only bulk-change fixVersion if you search only one project at a time. This URL may work - but edit the fixVersion part, and change LUCENE to SOLR to get to Solr's issues separately - https://issues.apache.org/jira/issues/?jql=project+=+LUCENE+AND+resolution=Unresolved+AND+fixVersion=6.0.1, and do a bulk change to the fixVersion to be both the master version and the next version on the branch you just released from. Uncheck the box that says "send an email for these changes".
4. Add a new Version for the next possible release version on the "Manage Versions" Administration page (https://issues.apache.org/jira/plugins/servlet/project-config/LUCENE/versions). e.g. If the current release is 5.2.1, add 5.2.2 with a description so that contributors can commit to the release branch with the next release version. In case of a minor release e.g. 5.2, this step needs to be done when the new release branch is cut.

Clear Security Level of Public JIRA Issues

ASF JIRA has a deficiency in which issues that have a security level of "Public" are nonetheless not searchable. Lets do a cleanup task now, even though it's not strictly release related:

1. It is a bit hard to find all Public issues as Security level is not in the dropdowns. So, the search query has to be in the advanced mode and be: project = SOLR AND Level = "Public"
2. Can only be done maximum 1000 issues at a time (under Tools/Bulk Change)
3. Check Edit
4. Set Security Level to None, scroll down all the way and uncheck Send email, then submit
5. Confirm
6. Wait (approx 7min per 1000)
7. Acknowledge

Stop mirroring old releases

Shortly after new releases are first mirrored, they are automatically copied to the archives. Only the latest point release from each active branch should be kept under the Lucene PMC svnpubsub area dist/releases/lucene/ and dist/releases/solr/. Older releases can be safely deleted, since they are already backed up in the archives.

Here's an example for the 6.0.1 release (note that the 5.5.1 release is not removed):

```
$ svn list https://dist.apache.org/repos/dist/release/lucene/java
5.3.1/
5.3.2/
5.4.1/
5.5.0/
5.5.1/
6.0.0/
6.0.1/
```

```
KEYS
README.html

$ svn rm -m "Stop mirroring old releases" https://dist.apache.org/repos/dist/release/lucene/java/
{5.3.1,5.3.2,5.4.1,5.5.0,6.0.0}

Committed revision 13826.

$ svn list https://dist.apache.org/repos/dist/release/lucene/solr
4.10.4/
5.2.0/
5.2.1/
5.3.0/
5.3.1/
5.3.2/
5.4.1/
5.5.0/
5.5.1/
6.0.0/
6.0.1/
HEADER.html
KEYS
ref-guide/

$ svn rm -m "Stop mirroring old releases" https://dist.apache.org/repos/dist/release/lucene/solr/
{4.10.4,5.2.0,5.2.1,5.3.0,5.3.1,5.3.2,5.4.1,5.5.0,6.0.0}

Committed revision 13827.
```

Update WIKI

The Solr WIKI has a page for every version which is often linked to from WIKI pages to indicate differences between versions, example: http://wiki.apache.org/solr/Solr4.3. Do the following:

1. Update the page for the released version with release date and link to release statement
2. Create a new placeholder page for the "next" version, if it does not exist

Personal notes

If you ran into any problems, possibly specific to your setup, you can add your notes here:

Ishan Chattopadhyaya's notes: 2017-02-28 https://docs.google.com/document/d/1Jys2mD2Y9ROOKuaxo7dZ457fYarHZcZJKHrLRJl20zo/edit#