Git commit process

Committing changes to Git

⚠️ Totally incomplete, but we have to start somewhere. Please contribute! ⚠️

Disclaimer Currently, we're still working out the preferred way of committing to Git, so this page is definitely a "work in progress". If you're a seasoned Git pro do not feel that what's written here is a requirement, but do please look it over and see if it's reasonably compatible.

There's been a lot of discussion about how to commit to the Git repo since we moved to using Git in early 2016. This page is intended for people new to Git or Lucene/Solr as a place to start. If you're totally new to Git or if you're a new committer, this should get you started.

NOTES:

- Only committers have the ability to commit to the official Git repository.
- IntelliJ and Eclipse have built-in Git integration
- There are visual tools that may be used as well. Mention here is not an endorsement, nor is it a complete list.
  - SourceTree Free.
  - SmartGit Licensed, not free. Well liked by at least some committers.

Useful Links

There are about a zillion "how to use Git" links out there, here are a few links that are useful in this context. Except for Dawid's and Apache's pages, these are not associated with this project.

- Git at Apache General Apache guidelines.
- Dawid's guide From our very own Dawid Weiss.
- Git for SVN crash course For people who know SVN but not Git.
- Let Me Google That For You I've wanted to paste a link to LMGTFY for a long time...

The goal here is to provide a super-simple guide without getting off into the all the possibilities, and it will certainly seem "too simple" to sophisticated users. We all have to start somewhere.

Git defaults

Here are some recommended defaults to configure Globally for Git:

```bash
  git config [--global] user.name <real-name>
  git config [--global] user.email <email>@apache.org

  #Is this really recommended?
  git config --global pull.rebase true
```

Simple commits

For simple commits for simple JIRAs for simple people, here are a few ways of going about it.

**method 1**

update your repo (perhaps with .gitconfig making this use rebasing)

- `git checkout origin/master`
- `Make changes, get through ant precommit and ant test targets`
- `git add`
- `git commit m="SOLR#### additional comments, often the title of the JIRA" (commits locally, nothing in ASF yet)`
- `git pull`
- `git push`

Notes:

- at any time `git status` tells you whether you have anything local that isn't in sync with the branch you're on. This includes changes not committed locally, new files not added to the local repository.
- `git diff --stat origin/master...` will show you committed (locally) but not pushed changes
- If you omit the --message flag you'll find yourself in a vi-like editor where you can enter long commit messages.

Backporting changes from trunk to 5x (6x sometime soon)

The usual recommendation is to "cherry pick".
• git checkout branch-5x
• git cherry-pick hash (You should have seen the hash echoed when you committed to origin/master, if not and you included the SOLR-####, the JIRA will have it)
• Check patch, perhaps run ant precommit and ant test targets again
• git commit (note this may already preserve the comments from the hash you’re cherry-picking from)
• git push

⚠️ There has been some issue with Solr's CHANGES.txt file "cherry picking" all of the changes for trunk, so check this file especially ⚠️

Add patches to JIRA tickets or not?
⚠️ what is the consensus here? ⚠️

More sophisticated options for more sophisticated people

• Dawid's Git Page Lots of options here, a fine way to get more sophisticated in your use of Git.

Working with Branches

Creating a Branch

Some feature work may be easier accomplished with a dedicated branch in Git. This allows several people to work together at once. Dawid's Git Page has details on how to create a branch and push it to the remote repository for others to pull and collaborate.

Preferred practice is to name your branch jira/solr-XXXX (where "solr-XXXX" is the JIRA ID), unless your feature does not yet have a single JIRA that's appropriate. In that case, you can use feature/<name>. If you name your branch in this way, commits to the branch will not pollute the comments of your JIRA issue.

Merging master with Your Branch

While working on your branch, you should periodically bring it up to date with "master". This will make eventual merging with master later simpler, because you resolve any conflicts as you go. To do this:

• git checkout master - first checkout master to be sure you have all the latest changes. If you are already on master, git pull to make sure you have the latest.
• git checkout <branch> - next checkout your branch.
• git merge master - this will apply all the changes in master that are not local to your branch.

At this point, you likely have conflicts. You need to resolve those, and there are a number of ways to do that.

Once the conflicts have been resolved:

• git commit - add your files. The commit message may be already populated with a message, edit it as needed.
• git push - push the merged branch to the remote repository.

Git may prompt you to git rebase --continue, if conflicts are resolved, that should end up in the same place, of prompting you to git push your final changes to the remote repository.

You may want to consider doing git merge --squash master - this will merge all commits from master into a single commit locally. Then when you commit the changes to your branch, the branch history won't become polluted with commits that didn't originate from the branch.

Merging Your Branch with master

When your feature is complete, and you want to merge the branch to master, you definitely want to do a squash merge so all of the little commits you made in the branch don't pollute the project master history.

You may also want to merge with master only after doing one last merge of master to your branch. This will allow you to resolve conflicts in your branch before having to deal with them on master, and will make this process cleaner.

Be sure you have run ant precommit on your branch to find any unexpected problems. Best to resolve those on your branch instead of waiting for Jenkins builds to fail, or other committers to notice problems. Running any tests you can is also preferred.

To merge your branch with master:

• git checkout <branch> - make sure you git pull to have the absolute latest changes from the remote repository.
• git checkout master - also, pull to make sure to git pull to have the absolute latest changes from the remote repository.
• git merge --squash <branch> - this will merge your branch with master.

Hopefully you have minimal or maybe no conflicts. If so, resolve them. When you're ready:

• git commit -m "Merging <branch> with master"

Before you push your changes to master, run ant precommit and tests again to check for anything unexpected.
• **git push**

That's it! Yeay!

**IntelliJ notes**

• At least one problem with IntelliJ (Community edition, version 15) is that it seems to get confused if you have both the SVN and Git plugins active at once. My (Erick Erickson) issues cleared up when I disabled the svn plugin.