Becoming A Nutch Developer

Overview

Search is complex. Nutch makes it easier. So you start off by installing Nutch. Now you are a pretty good developer. You get Nutch up and running without problems and you think it is a pretty neat piece of software. In fact you like it so much that you want to start adding to it. You want to develop new features and contribute them back to the community. But then it dawns on you. How does one contribute to an open source project like Nutch. You have never worked on an open source project before and don't really know how the entire process works.

That is where this document comes in. The purpose of this document is to help you as developers take the next step in becoming contributing members of the Nutch community. We will cover a general overview of the Nutch development process including the different pieces and how they fit together. We will cover how the community works and interacts, using the mailing lists to search for information and how to ask questions to ensure that they get answered. We will also cover how to go about learning the internals of the Nutch codebase. We will cover how to use the JIRA for change requests and how to start developing for Nutch. And finally we will cover contributing back to the Nutch community. When we are finished you should have a good understanding of how the community works and how you can go about becoming a bigger part of that community.

Table of Contents

The Nutch Community

Nutch Development Roles

There are three main roles that a person can play in the Nutch community.

The first role is that of user. This is someone who uses the Nutch software but is not active in its development. People in this category range from the curious programmer who wants to learn more about search technology to corporations setting up search on their local intranet. If you only want to use the Nutch software and don't want to help develop it, you can still be a contributing member of the community. By using the software and pushing the limits of what it can do, by filing bug reports and feature requests (more about how to do this later), by working with developers to track down issues, or just giving your input to discussions that arise, you can help the Nutch project become better and better.

The second role is that of developer. This is someone who has used the Nutch software and has taken the next step to help program the underlying software. Since you are reading this document, it is assumed that you want to be a developer. Helping to develop the Nutch codebase can come in the form of bug fixes or by developing completely new features from scratch. An important thing to remember is that unlike most software development at big companies, you don't need anyone's permission to start developing software for Nutch. If you think you have a good idea for a feature, or if you want to track down and fix bugs in the software, go for it. If you want a specific piece of functionality, don't wait for someone else to develop it. Take the time to learn and develop it yourself. Then when you are done give it back to the community. This is how the Nutch project has been developed so far and how it will continue to be developed in the future. The community is a do-ocrcy, meaning those who do the work get to help set the directions and make the decisions. Communication is essential but not limiting. Anybody can become a developer simply by taking the initiative, whether in the form of fixes or functionality, for inclusion in the project.

The third role is that of committer. This is usually a developer who has been working with the project for some time. Someone who has developed new pieces of functionality, who has fixed bugs, and who has helped others through answering questions and providing guidance to others through the mailing lists and wiki. In other words this is a person who has proved their commitment and usefulness to the project and in return are given commit access to the source code repository, an apache email address, and the ability to help make strategic decisions for the project by determining what submissions and bug fixes make it into the source code repository and release versions of the software.

How the Community Works

The community works together through shared mailing lists, email, wikis, bug tracking systems, and source code repositories. These tools when used together provide a virtual meeting room and workspace for all members of the community.

Mailing Lists and Email

Most communication is done through email and the mailing lists. Because of this the first thing that any person should do to become part of the Nutch community is to join the appropriate mailing lists. There are four different mailing lists. First there is the users mailing list. Contrary to the name this list is not just for users. If you have questions about the Nutch software including installation, configuration, bugs, errors, or general information, this is the list for you. Second is the dev mailing list. This is where most development communication occurs including updates to request tracking systems. This is also where developers can pose ideas for new functionality to see if someone is already working on such features or just to get general feedback. The dev mailing list is important for tracking functionality that other developers may be working on and consensus by the community on desired direction of new features. The third list is the commit mailing lists. This list tracks commits to the source code repository and changes in the wiki pages. The fourth list is the agents list. This is where webmasters and other people can post comments or questions about the Nutch crawler.

Users can get by with subscribing to only the users mailing list. Developers should subscribe to all four mailing lists. Anybody doing internet crawls needs to be subscribed to the agents list. In order to post to any list, for example to ask a questions, it is necessary to first be subscribed to that list. Below are links for subscribing to the different mailing lists.

- Nutch Mailing Lists
- Subscribe to Users
- Subscribe to Dev
- Subscribe to Commits
- Subscribe to Agents

JIRA and Issue/Request Tracking

If mailing lists provide the ongoing conversation for the community, the issue/request tracking system provides a repository for the current state of the project. The request tracking system is JIRA system and it can be accessed at this address.

- Nutch JIRA
- Signup for JIRA

The JIRA system is the central repository for all work wanting to be included int the Nutch source code base. The system tracks issues and feature requests by component, by version, and by status. You can view what requests are assigned to what person, what requests are currently being worked on, and which ones haven't been scheduled. You can search all requests by keyword or by various categories and filters. We will go into detail later on how to use the JIRA system to propose new functionality and submit bug fixes. For now understand this: If you are going to be a developer you will need to understand how to use the JIRA system as this is where you will propose new functionality, submit bug fixes, give you input on features other developers may be working on, and coordinate actions with other developers on specific pieces of functionality.

The address to signup for JIRA was given above. Once you have signed up you will have access to all of the Apache JIRA repository, not just the Nutch project.

Source Code Control through Git

Source code control is very important to open source projects. Nutch uses Git for its source control. See UsingGit how to access the Nutch source repositories.

Wiki and Documentation

The weakest part of most open source projects is their documentation and Nutch is no exception. Wikis are special web pages like the one that you are reading that allows users to directly edit text on the page and to create new pages. The wiki provides various tutorials and documentation for Nutch. Links to view the Nutch wiki and to register for the wiki are provided below.

- Nutch Wiki
- Signup for the Wiki

As a developer one of the ways you can contribute back to the community is by documenting your hard won experience on the wiki. You can do this in the form of tutorials, articles, or simple notes and instructions. Anything that you have learned may be of use to other developers. The wiki is also used as a virtual white board to help document general themes and directions for the project.

These four tools, mailing lists and email, JIRA, Wikis, and Git together provide the community ways to coordinate their actions and conversations. As a developer you will need to understand each of these tools and how you will use them in the development process.

Next Steps

The rest of this document will cover four steps in becoming a Nutch developer. First is using the mailing lists to communicate, find information, and get solutions to problems. Second is how to go about learning the different parts of the Nutch codebase. Third is how to use the JIRA to search for bugs and coordinate development efforts. And four is how to use the wiki to help grow the community knowledge base.

Becoming a Nutch Developer

Step One: Using the Mailing Lists

The most used tool in Nutch development is by far the mailing lists. We have already explained the various mailing lists and their uses. This section will provide general guidance on using the mailing lists to find information and get questions answered.

First and foremost, any person wanting to become a Nutch developer should start reading the user, dev, and commits lists on a daily basis. To start out with simply read the questions that other users ask on the users list. As you begin delving into the Nutch codebase more and more you will be able to answer some of the questions that other users have. One of the best ways to learn Nutch is by daily taking one question that is asked on the users list and seeing if you can find the answer.

As a developer you will want to keep up with the current state of development on the project and this is where the dev list comes in. The dev list is where JIRA messages will be delivered every time a JIRA request is updated. By following this list you will see discussions between other developers about various bugs and feature requests. You may also see an information voting occuring. If you feel you can contribute to one of the discussions, by all means add your input. You will also want to keep up to date on the commits list to see what code has been committed to the git repository and what updates have been made to the wiki.

Don't think that you have to be an expert on Nutch to begin answering questions. If you think you can help another user on the mailing list, don't be afraid to go and do it. I think that it was Eric Raymond that said "Given enough eyeballs, all bugs are shallow". What this means is that you bring a unique perspective to the world as does everyone else and you may find bugs that no one else can find. The more people we have looking at the code and improving upon it, the more stable and robust it will become. The more people we have in the community constantly communicating, asking questions and helping each other, the better we all become.

As you start to have questions about the configuration and operation of Nutch or about errors that you have recieved, go ahead and ask these questions on the user list. When asking questions it is best to provide a descritive subject and detailed information about the problem or question. Detailed information would include snippets of log or configuration files and a good description of the problem. In general the more specific information you can provide, the easier it is for other users and developers to help. General or abstract questions tend to be ignored. For example I have seen messages on the list like this before that were completely ignored.

A Bad Email

```
Subject: Problem with crawling

I am having a problem with crawling the internet.  It just seems that it is taking a long time.  Does anyone
know why crawling takes so long.

Me
```

With this type of question other users would have no idea what the problem is or how to help and therefore most simply ignore the question and move on. On the other hand here is better example of asking questions.

A Good Email

```
Subject: Crawl on 20K pages taking 4 hours

I am using Nutch .8 branch over a cluster of 3 machines each running redhat linux and java 1.5_10 with 500G
hard drives, 2.8 Ghz processors, and 2G of ram.  I am trying to fetch 20K urls and the fetch process completes
fine but when it gets to the reduce process, the cpus go to 100% and the process seems to spin indefinitely.  I
did a kill -SIGQUIT on the process and it seems to be stuck on the Regex normalizer class.  Has anyone
experienced similar problems or know what might be causing this problem.

Me
```

In the second case, much more detailed information such as operating system, java version, component in which the error is occurring, and a detailed description of the problem, is provided and developers are much better equipped to provide assistance.

Before asking questions on the list you will want to search the list to see if your problem has come up in the past. There may already be a solution to your problem out there. I have seen times when questions went unanswered on the list because the same question had been answered only a few days before and the person asking never bothered to search the archives of the list. Below are two different web based locations from which you can search the Nutch mailing lists.

- Nutch Mail Archive
- Nabble Nutch
- Lucid Imagination Email

When searching the list for errors you have received it is good to search both by component, for example fetcher, and by the actual error received. If you are not finding the answers you are looking for on the list, you may want to move to the JIRA and search there for answers.

Here are some other important things to remember about the mailing lists. First, do not cross post questions. Find the best list for your question and post it to that list only. Posting the same question to multiple lists (i.e. user and dev) tends to annoy the very people you want help from. Second, remember that developers and committers have day jobs and deadlines also and that being rude, offensive, or aggressive is a sure way to get your posting ignored if not flamed.

Most questions on the lists are answered within a day. If you ask a question and it is not answered for a couple of days, do not repost the same question. Instead you may need to reword your question, provide more information, or give a better description in the subject.

Step Two: Learning the Nutch Source Code

I have found that when teaching new developers the basics of the Nutch source code it is easiest to first start with learning the operations of a full crawl from start to finish.

A word about Hadoop. As soon as you start looking into Nutch code (versions .8 or higher) you will be looking at code that uses and extends Hadoop APIs. Learning the Hadoop source code base is as big an endeavor as learning the Nutch codebase, but because of how much Nutch relies on Hadoop, anyone serious about Nutch develop will also need to learn the Hadoop codebase.

First start by getting Nutch up and running and having completed a full process of fetching through indexing. There are tutorials on this wiki that show how to do this. Second get Nutch setup to run in an integrated development environment such as Eclipse. There are also tutorials that show how to accomplish this. Once this is done you should be able to run individual Nutch components inside of a debugger. This is essential because probably the fastest way to learn the Nutch codebase is to step through different components in a debugger.

Start by looking at the crawl package, specifically the Injector, Generator, and CrawlDb classes. Start with Injector, read over the source code for it and classes that it references. Run the class as a main program inside of a debugger. You will want to do this in local mode on the local file system.

You will also want to take a look at the junit test classes for the same package. These can be found under the src/test folder. There should be test classes for all of the main components. Again read the source code for the test classes and execute the junit test cases to get a deeper understand of how each component works.

Follow this pattern of reading the source code and and classes it references, running through the source in a debugger if possible, and reading and executing the junit test cases for each component in the crawl process. In order they are Injector, Generator, Fetcher, ParseSegment, CrawlDb, LinkDb, Indexer, DeleteDuplicates. Then you will want to look at other components including SegmentMerger, CrawlDbMerger, and DistributedSearch. It is usually best to get through one or two components before beginning to look at he Hadoop code and then switch back and forth between the Nutch and Hadoop code to understand where and how Nutch uses Hadoop.

In Hadoop it is better to take the packages one at a time, for example mapred or dfs, then to take the strategy of running components as Hadoop is server based. You can still follow the pattern of reviewing junit tests to get an understanding of the Hadoop source code. Once you feel you have a grasp of various parts of the source code in Nutch or Hadoop I would recommend creating small junit test cases that use your newfound knowledge. For example you can create a small test case that fetches a few urls and verifies that they were fetched correctly. If you get through all of this then you will have a good foundation of knowledge in the Nutch and Hadoop source code bases and you should fine in starting to develop software for both Nutch and Hadoop.

Step Three: Using the JIRA and Developing

Ok, so you have gone through the source code and have a good understand of the different components. Now you want to start developing or fixing bugs. Where do you start. First if you haven't already signed up for the JIRA, do so now. Instructions were provided earlier for this.

Now it is time to start browsing. JIRA provides a lot of search facilities. On the top of the main JIRA page there is a free text search. On the right hand side of the main JIRA page there are preset filters. You can search by status of the issue, by priority, or by assignee. You will want to try out each of the different search options to get familiar with the capabilities of JIRA.

When you do a search in JIRA you are presented with a listing of issues that match your query. The results listing will show you the JIRA id for the Nutch issue. This is in the form of NUTCH-XXX. It is important to remember the JIRA id numbers as this is how you will reference issues that you are working on both through the JIRA and in communicating with other developers on the list. The listing also shows a brief summary of the issue, who it is assigned to, who reported it, the priority and status of the issue, and if it is resolved. Clicking on the issue number will bring you to the main page for that issue. The main issue page is where you will communicate with other developers about this issue and where you will attach your code patches for bug fixes and new feature requests. Whenever changes are made to JIRA issues an email is automatically generated and sent to the dev mailing list. You will have to be logged in to leave comments or to attach documents to issues. Again it is important to become familiar with the interface.

Once you have become familiar with the JIRA interface it is time to pick something to work on. If you already have something that you wish to work on, either a bug fix or a new piece of functionality then the first step is to send a message to the dev mailing list detailing the issue. By doing this you can get feedback from other developers. You may find that someone is already working on the issue or that the functionality is handled elsewhere. Either way first notifying the list, especially if it a major piece of new functionality, is the polite thing to do.

On a side not, before you start creating issues in the JIRA that you are going to work on yourself you need to send an email to the developers list asking to be added to the nutch-developers group. Then when you create issues later in the JIRA you can have the issues you create assigned to you. This helps other developers know what is being worked on at any given point in time and avoids duplication of effort.

Once you have gotten feedback from other developers and no one has objected then you will need to create an issue in the JIRA. In the JIRA issue please give as much detail and description as possible. Once the issue is created assign the issue to yourself or if you don't have permissions to do so then send a message to the dev mailing list asking that the issue be assigned to you.

If you are not creating an new issue but instead want to begin working on an existing issue then here are the steps. First find the issue that you want to work on. If it is assigned to someone else then send a message to that person to see if they are working on it and where they are at in their process. It often happens that issues get assigned to developers but the developers are too busy to work on them. Or it may be that the person is in the process of working on the issue and would welcome your help. Either way, you should always contact that person and coordinate the efforts. That's only polite and sensible.

Regarding the picking of the work to be done - natural ordering in JIRA should be followed. Issues marked critical are more important than "major", and the ones with a lot of votes are more important than those without any.

Once the JIRA is created and has been assigned to you then it is time to start coding. Remember to follow a few simple guidelines while coding.

- All public classes and methods should have informative Javadoc comments.
- Code should be formatted according to Sun's conventions, with one exception indent two spaces per level, not four. For Eclipse the code style formatter profile defined in eclipse-codeformat.xml can be used to automatically format (only, see below) the contributed code.
- Contributions should pass existing unit tests.
- New junit test cases should be provided to demonstrate bugs fixes and new features.

You will also want to perform functional testing of your new code within your own environment as well as make sure that the ant build and javadoc are successful with your new code. Once your code has been completed and tested then it is time to create a patch.

Start by checking to see what files you have modified with:

```
git status
```

Keep this list for later because you will want to make sure that only code that you have changed is included in your patch.

In order to create a patch, just type:

```
git diff > yourPatchName.patch
```

This will report all modifications done on the Nutch sources on your local disk and save them into the yourPatchName.patch file. Read the patch file. Make sure it includes ONLY the modifications required to fix a single JIRA issue. More information on creating patches is provided in HowToContribute.

Please do not:

- reformat code unrelated to the bug being fixed: formatting changes should be separate patches/commits.
- comment out code that is now obsolete: just remove it.
- insert comments around each change, marking the change: folks can use git to figure out what's changed and by whom.
- make things public which are not required by end users.

Please do:

- try to adhere to the coding style of files you edit.
- comment code whose function or rationale is not obvious.
- update documentation (e.g., package.html files, this wiki, etc.)

Finally, patches should be attached to the JIRA issue. You can do this by logging into the JIRA issue and clicking the attach file to this issue link on the left hand side of the JIRA issue page.

Then please be patient and in the mean time start working on another issue. Committers are busy people too. If no one responds to your patch after a few days, please make friendly reminders to the dev mailing list. Please incorporate other's suggestions into into your patch if you think they're reasonable.

Now here is the hard part. Even if you have completed your patch it may not make it into the final Nutch codebase. This could be for any number of reason but most often it is because the piece of functionality is not in lines with the strategic goals of Nutch. Of course if you had sent an email to the list before starting development on the issue then this would have already been addressed. Remember though that all developers have access to your functionality through the JIRA and they can and will use your patch even if it does not make it into release code. Every patch is useful to the community.

Step Four: Contributing

This is the easy step. As you get more and more understanding in the Nutch code base. It is useful to take your hard earned knowledge and start helping others in the community. You can do this by creating tutorials, articles, and notes on the wiki, by answering questions on the mailing lists, or by testing and reviewing patches. Remember that the project is a circle. The more people you help the better they become and better functionality they develop that in turn helps you. Together we can all lift each other higher.

Becoming a Nutch Committer

So you have developed some very useful functionality and contributed it back to the community. You consistently fix bugs. You answer questions for other users and developers on the mailing lists. All in all you are an asset to the community. At this point you may be invited to become a committer. At this point you would get an apache email address and direct access to the git source code repository and you would be responsible for helping set the technical direction of the Nutch project.

Conclusion

So I hope this tutorial has helped to guide you in the direction of becoming a Nutch developer. Nutch is an awesome piece of software that has tremendous potential for changing search as we know it. If you desire to work on a piece of software that has the potential to affect millions of people around the world, then this is the project for you. Get started today and in a year or two you will look back and be amazed at just how much you have accomplished.

I would like to thank Andrzej Bialecki, Chris Mattmann, and Doug Cutting for providing assistance in developing this tutorial. I hope to meet, as we say it in Texas, ya'll in person one day.