Nutch version 0.8 tutorial

Table of contents

1. Requirements

1. Java 1.4.x, either from Sun or IBM on Linux is preferred. Set `NUTCH_JAVA_HOME` to the root of your JVM installation.
2. Apache's Tomcat 4.x.
3. On Win32, cygwin, for shell support. (If you plan to use Subversion on Win32, be sure to select the subversion package when you install, in the "Devel" category.)
4. Up to a gigabyte of free disk space, a high-speed connection, and an hour or so.

2. Getting Started

First, you need to get a copy of the Nutch code. You can download a release from http://lucene.apache.org/nutch/release/. Unpack the release and connect to its top-level directory. Or, check out the latest source code from subversion and build it with Ant.

Try the following command:

```
bin/nutch
```

This will display the documentation for the Nutch command script.

Now we're ready to crawl. There are two approaches to crawling:

1. Intranet crawling, with the `crawl` command.
2. Whole-web crawling, with much greater control, using the lower level `inject`, `generate`, `fetch` and `updatedb` commands.

3. Intranet Crawling

Intranet crawling is more appropriate when you intend to crawl up to around one million pages on a handful of web servers.

3.1. Intranet: Configuration

To configure things for intranet crawling you must:

1. Create a directory with a flat file of root urls. For example, to crawl the `nutch` site you might start with a file named `urls/nutch` containing the url of just the Nutch home page. All other Nutch pages should be reachable from this page. The `urls/nutch` file would thus contain:

   ```
   http://lucene.apache.org/nutch/
   ```

2. Edit the file `conf/crawl-urlfilter.txt` and replace `MY.DOMAIN.NAME` with the name of the domain you wish to crawl. For example, if you wished to limit the crawl to the `apache.org` domain, the line should read:

```
+^http://([a-z0-9]*\.)*apache.org/
```

This will include any url in the domain `apache.org`.

3.2. Intranet: Running the Crawl

Once things are configured, running the crawl is easy. Just use the crawl command. Its options include:

- `-dir` *dir* names the directory to put the crawl in.
- `-threads` *threads* determines the number of threads that will fetch in parallel.
- `-depth` *depth* indicates the link depth from the root page that should be crawled.
- `-topN` *N* determines the maximum number of pages that will be retrieved at each level up to the depth.

For example, a typical call might be:

```
bin/nutch crawl urls -dir crawl -depth 3 -topN 50
```

Typically one starts testing one's configuration by crawling at shallow depths, sharply limiting the number of pages fetched at each level (`-topN`), and watching the output to check that desired pages are fetched and undesirable pages are not. Once one is confident of the configuration, then an appropriate depth for a full crawl is around 10. The number of pages per level (`-topN`) for a full crawl can be from tens of thousands to millions, depending on your resources.

Once crawling has completed, one can skip to the Searching section below.

4. Whole-web Crawling

Whole-web crawling is designed to handle very large crawls which may take weeks to complete, running on multiple machines.

4.1. Whole-web: Concepts

Nutch data is composed of:

1. The crawl database, or *crawldb*. This contains information about every url known to Nutch, including whether it was fetched, and, if so, when.
2. The link database, or *linkdb*. This contains the list of known links to each url, including both the source url and anchor text of the link.
3. A set of *segments*. Each segment is a set of urls that are fetched as a unit. Segments are directories with the following subdirectories:
4. • a *crawl_generate* names a set of urls to be fetched

- a *crawl_fetch* contains the status of fetching each url
- a *content* contains the content of each url
- a *parse_text* contains the parsed text of each url
- a *parse_data* contains outlinks and metadata parsed from each url
- a *crawl_parse* contains the outlink urls, used to update the crawldb

5. The *indexes*are Lucene-format indexes.

4.2. Whole-web: Boostrapping the Web Database

The *injector* adds urls to the crawldb. Let's inject URLs from the DMOZ Open Directory. First we must download and uncompress the file listing all of the DMOZ pages. (This is a 200+Mb file, so this will take a few minutes.)

```
wget http://rdf.dmoz.org/rdf/content.rdf.u8.gz
gunzip content.rdf.u8.gz
```

Next we select a random subset of these pages. (We use a random subset so that everyone who runs this tutorial doesn't hammer the same sites.) DMOZ contains around three million URLs. We select one out of every 5000, so that we end up with around 1000 URLs:

```
mkdir dmoz
bin/nutch org.apache.nutch.crawl.DmozParser content.rdf.u8 -subset 5000 >
dmoz/urls
```

The parser also takes a few minutes, as it must parse the full file. Finally, we initialize the crawl db with the selected urls.

```
bin/nutch inject crawl/crawldb dmoz
```

Now we have a web database with around 1000 as-yet unfetched URLs in it.

4.3. Whole-web: Fetching

To fetch, we first generate a fetchlist from the database:

```
bin/nutch generate crawl/crawldb crawl/segments
```

This generates a fetchlist for all of the pages due to be fetched. The fetchlist is placed in a newly created segment directory. The segment directory is named by the time it's created. We save the name of this segment in the shell variable `s1`:

```
s1=`ls -d crawl/segments/2* | tail -1`
echo $s1
```

Now we run the fetcher on this segment with:

```
bin/nutch fetch $s1
```

When this is complete, we update the database with the results of the fetch:

```
bin/nutch updatedb crawl/crawldb $s1
```

Now the database has entries for all of the pages referenced by the initial set.

Now we fetch a new segment with the top-scoring 1000 pages:

```
bin/nutch generate crawl/crawldb crawl/segments -topN 1000
s2=`ls -d crawl/segments/2* | tail -1`
echo $s2

bin/nutch fetch $s2
bin/nutch updatedb crawl/crawldb $s2
```

Let's fetch one more round:

```

bin/nutch generate crawl/crawldb crawl/segments -topN 1000
s3=`ls -d crawl/segments/2* | tail -1`
echo $s3

bin/nutch fetch $s3
bin/nutch updatedb crawl/crawldb $s3
```

By this point we've fetched a few thousand pages. Let's index them!

4.4. Whole-web: Indexing

Before indexing we first invert all of the links, so that we may index incoming anchor text with the pages.

```
bin/nutch invertlinks crawl/linkdb crawl/segments
```

To index the segments we use the `index` command, as follows:

```
bin/nutch index indexes crawl/linkdb crawl/segments/*
```

Now we're ready to search!

4.5. Searching

To search you need to put the nutch war file into your servlet container. (If instead of downloading a Nutch release you checked the sources out of SVN, then you'll first need to build the war file, with the command `ant war`.)

Assuming you've unpacked Tomcat as ~/local/tomcat, then the Nutch war file may be installed with the commands:

```
rm -rf ~/local/tomcat/webapps/ROOT*
cp nutch*.war ~/local/tomcat/webapps/ROOT.war
```

The webapp finds its indexes in `./crawl`, relative to where you start Tomcat, so use a command like:

```
~/local/tomcat/bin/catalina.sh start
```

Then visit http://localhost:8080/ and have fun!

More detailed tutorials are available on the Nutch Wiki.