In [51]:
```
import nltk
import urllib2
import re
```

Automatic Tagging with NLTK

Although the above results are neat, they aren't all that useful in practice because most texts we want to visualize in such ways aren't tagged, and tagging them by hand ist costly.

What we need is an *automated tagger*.

Let's take a page off Wikipedia and tag it automatically.

In [52]:
```
opener = urllib2.build_opener()
opener.addheaders = [('User-agent', 'Mozilla/5.0')]
infile = opener.open('http://en.wikipedia.org/w/index.php?title=George_Washington&pr
page = infile.read().decode("utf-8")
page[:400]
```

Out[52]:
```
u'<!DOCTYPE html>\n<html lang="en" dir="ltr" class="client-nojs">\n<head>\n<title>G
Washington - Wikipedia, the free encyclopedia</title>\n<meta charset="UTF-8" />\n<m
name="generator" content="MediaWiki 1.21wmf4" />\n<meta name="robots" content="noin
/>\n<link rel="apple-touch-icon" href="//en.wikipedia.org/apple-touch-icon.png" />\
rel="shortcut icon" href="/favicon.ico" />\n<link '
```

This is in HTML format, so we first need to clean it up.

(There are other ways of cleaning up and analyzing HTML. A good HTML library is BeautifulSoup.)

In [53]:
```
page = nltk.util.clean_html(page)
page = re.sub(r'\s*\n+','\n',page)
print page[:400]
```

```
George Washington - Wikipedia, the free encyclopedia


encyclopedia
                                                Jump to:
,                                                search

This article is about the first President of the United States. For other uses, see
 For a simpler version of this article, see the Simple English Wikipedia article:
```

All the rest of the software works on tokenized data.

In [54]:
```
sents = nltk.sent_tokenize(page)
sents = [nltk.word_tokenize(s) for s in sents]
print sents[17]
```

```
[u'Washington', u'quickly', u'became', u'a', u'senior', u'officer', u'in',
u'the', u'colonial', u'forces', u'during', u'the', u'first', u'stages', u'of',
u'the', u'French', u'and', u'Indian', u'War', u'.']
```

This data hasn't been manually tagged, so we need an automatic tagger.

In [55]:
```
print nltk.pos_tag(sents[17])
```

```
[(u'Washington', 'NNP'), (u'quickly', 'RB'), (u'became', 'VBD'), (u'a', 'DT'),
(u'senior', 'JJ'), (u'officer', 'NN'), (u'in', 'IN'), (u'the', 'DT'),
(u'colonial', 'JJ'), (u'forces', 'NNS'), (u'during', 'IN'), (u'the', 'DT'),
(u'first', 'JJ'), (u'stages', 'NNS'), (u'of', 'IN'), (u'the', 'DT'), (u'French',
'JJ'), (u'and', 'CC'), (u'Indian', 'JJ'), (u'War', 'NN'), (u'.', '.')]
```

To perform named entity extraction, we run `ne_chunk` on the output of `pos_tag`.

The output is a mix of tree nodes combining multiple tagged words, together with raw tagged words.

In [56]:
```
chunked = nltk.ne_chunk(nltk.pos_tag(sents[17]))
for node in chunked:
    print node
```

```
(GPE Washington/NNP)
(u'quickly', 'RB')
(u'became', 'VBD')
(u'a', 'DT')
(u'senior', 'JJ')
(u'officer', 'NN')
(u'in', 'IN')
(u'the', 'DT')
(u'colonial', 'JJ')
(u'forces', 'NNS')
(u'during', 'IN')
(u'the', 'DT')
(u'first', 'JJ')
(u'stages', 'NNS')
(u'of', 'IN')
(u'the', 'DT')
(GPE French/JJ)
(u'and', 'CC')
(GPE Indian/JJ)
(u'War', 'NN')
(u'.', '.')
```

There are actually several different kinds of named entities.

In [57]:
```
"""
ORGANIZATION    Georgia-Pacific Corp., WHO
PERSON          Eddy Bonte, President Obama
LOCATION        Murray River, Mount Everest
DATE            June, 2008-06-29
TIME            two fifty a m, 1:30 p.m.
MONEY           175 million Canadian Dollars, GBP 10.40
PERCENT         twenty pct, 18.75 %
FACILITY        Washington Monument, Stonehenge
GPE             South East Asia, Midlothian
""";None
```

We now need to dig through this data to extract the actual information about the named entity.

In [58]:
```
def nextract(tokens,types=["GPE","PERSON"]):
    chunked = nltk.ne_chunk(nltk.pos_tag(tokens))
    return [c for c in chunked if hasattr(c,"node") and c.node in types]
nes = nextract(sents[17])
nes
```

Out[58]:
```
[Tree('GPE', [(u'Washington', 'NNP')]),
 Tree('GPE', [(u'French', 'JJ')]),
 Tree('GPE', [(u'Indian', 'JJ')])]
```

In [59]:
```
nes[0].leaves()
```

Out[59]:
```
[(u'Washington', 'NNP')]
```

In [60]:
```
def nextract_text(tokens,types=["GPE","PERSON"]):
    nodes = nextract(tokens,types)
    return [" ".join(c[0] for c in chunk.leaves()) for chunk in nodes]
```

In [61]:
```
nextract_text(sents[17])
```

Out[61]:
```
[u'Washington', u'French', u'Indian']
```

In [62]:
```
nes = [nextract_text(s,["PERSON"]) for s in sents]
```

Let's look at what this extracted.

In [49]:
```
from collections import Counter
Counter([x for l in nes for x in l]).most_common(10)
```

Out[49]:
```
[(u'George Washington', 90),
 (u'George', 44),
 (u'Mount Vernon', 22),
 (u'Martha', 14),
 (u'Jefferson', 8),
 (u'Chernow', 8),
 (u'John Adams', 8),
 (u'See', 7),
 (u'Oxford University', 6),
 (u'John', 6)]
```

As you can see, the named entity extractor has a significant error rate. "Mount Vernon", "See", and "Oxford University" are not persons. Also, we don't know the identity of "George", "Martha", and "John". But generally, it seems to return the right thing.

Now let's look at co-occurences of named entities.

In [50]:
```
from itertools import *
pairs = Counter([tuple(sorted(list(p))) for s in nes for p in combinations(s,2)])
[p for p in pairs.most_common(20) if p[0][0]!=p[0][1]]
```

Out[50]:
```
[((u'George Washington', u'Mount Vernon'), 17),
 ((u'George Washington', u'Lawrence Washington'), 12),
 ((u'John Adams', u'Position'), 10),
 ((u'George Washington', u'Henry Knox'), 8),
 ((u'George Washington', u'Henry Compton'), 8),
 ((u'George Washington', u'Martha Washington'), 8),
 ((u'George Washington', u'John Adams'), 8),
 ((u'George Washington', u'William Wake'), 8),
 ((u'George Washington', u'John Tyler'), 8),
 ((u'George Washington', u'Gibson'), 8),
 ((u'George Washington', u'William &amp'), 8),
 ((u'George Washington', u'Timothy Pickering'), 8),
 ((u'Gardens Discover', u'George Washington'), 7),
 ((u'George Washington', u'George Washington Birthplace National Monument'),
  7),
 ((u'George Washington', u'Mount Vernon Estate'), 7),
 ((u'George Washington', u'Miller Center'), 7),
 ((u'George Washington', u'Museum &amp'), 7),
 ((u'George Washington', u'Made George Washington'), 7)]
```

In [ ]: