# Auto-Archive Documentation

## How it works:

The code I implemented archives the current state of a page before each edit. It does this by simply copying (copy() from shutil) the file to another directory and changing its filename and date: metadata tag. I added the date: tag in each markdown creation. The revisions for each file can be found using the “Revisions” button on the sidebar.

## Modified files:

The following source code files were modified:

Core.py Line 7 and 11 Imports to handle date and copy

from datetime import date, datetime, time

from shutil import copyfile, copy, copy2

Lines 195-197 Make archive if file is being edited

# makes an archive only if page exists already  
 if os.path.isfile(self.path):  
 self.archive(folder)

Lines 208-228 Copy file to directory, rename it, and change the date

def archive(self, folder):  
 *"""  
  
 Copies file before edit into the "revised/url" folder  
 and gives it a unique name based on the current time  
 folder will be created if it's the first edit and will persist  
 even if the title is changed  
  
 """* revised\_folder = folder + "/revised/" + self.url  
 timestr = datetime.now().strftime("%Y%m%d-%H%M%S")  
 x = datetime.now()  
 if not os.path.exists(revised\_folder):  
 os.makedirs(revised\_folder)  
 copy2(self.path, revised\_folder)  
 new\_file = revised\_folder + "/" + self.url + "\_" + timestr + ".md"  
 os.rename(revised\_folder + "/" + self.url + ".md", new\_file)  
 with open(new\_file, 'r+', encoding='utf-8') as f:  
 data = f.read()  
 f.seek(data.index("date: "))  
 f.write("\ndate: " + x.strftime("%c") + '\n')

and lines 365-383 Mirrors index() but finds all files in content/revised/url

def revisions(self, url):  
 *"""  
 Similar to index, builds a list of all the available revisions of a page* ***:returns****: a list of all revisions* ***:rtype****: list  
 """* pages = []  
 root = os.path.abspath(self.root)  
 for cur\_dir, \_, files in os.walk(root + "/revised/" + url):  
 # get the url of the current directory  
 cur\_dir\_url = cur\_dir[len(root)+1:]  
 for cur\_file in files:  
 path = os.path.join(cur\_dir, cur\_file)  
 if cur\_file.endswith('.md'):  
 url = clean\_url(os.path.join(cur\_dir\_url, cur\_file[:-3]))  
 page = Page(path, url)  
 pages.append(page)  
 return sorted(pages, key=lambda x: x.title.lower())

routes.py Line 23 import to handle date

from datetime import datetime

Lines 70-83 Close to index(url) but handles revisions instead

"""  
  
revisions() requires only  
  
from datetime import datetime  
  
will render a page with a list of revisions from content/revisions/url  
  
"""  
@bp.route('/revisions/<path:url>/', methods=['GET', 'POST'])  
@protect  
def revisions(url):  
 pages = current\_wiki.revisions(url)  
 return render\_template('revisions.html', pages=pages)

Line 98 Pass date to editor so we can add it to the markdown

return render\_template('editor.html', form=form, page=page, date=datetime.now().strftime("%c"))

Forms.py Line 44 Add new input field in EditorForm to handle the date

date = wtforms.TextField('', render\_kw={'readonly': True})

Editor.html Line 17 Date field

{{ input(form.date, placeholder=date, class="span7", autocomplete="off", value=date) }}

Page.html Line 4-6, 30, 44, 46, and 49 All of these handle the page differently if it is an archived file

{% if "revised" **in** page.path %}  
 <p style="font-size:15px">Edited on {{ page.date }}</p>  
{% endif %}

{% if "revised" **not in** page.path %}

{% endif %}

{% if "revised" **not in** page.path %}

{% endif %}

Revisions.html is a new file, it mirrors index.html, but displays revisions for a page with title, url, and date.

# Design Manual

## Problem definition

Edits are final, and there was previously no way to go back and view previous states of the page. This makes it to where if you forget something that was on a page, or missed it, you can’t go back and look at the previous state. In order to solve the problem, we can archive pages on every edit.

## Basic Requirements

* Save pre-edit page in an archive in the correct directory
* Be able to view all previous revisions of a page

## User stories

* As a wiki admin, I would like to see what has been edited, and when it was edited
* As a wiki user, I would like to see changes for this page as I use it frequently for information, and do not want to be out of the loop.

## Design/Implementation plan

The feature will archive a page automatically, an easy way to do this would be to copy the file before it’s edited, give it a unique name from the current time, and change the date in the markdown file

## Test plan

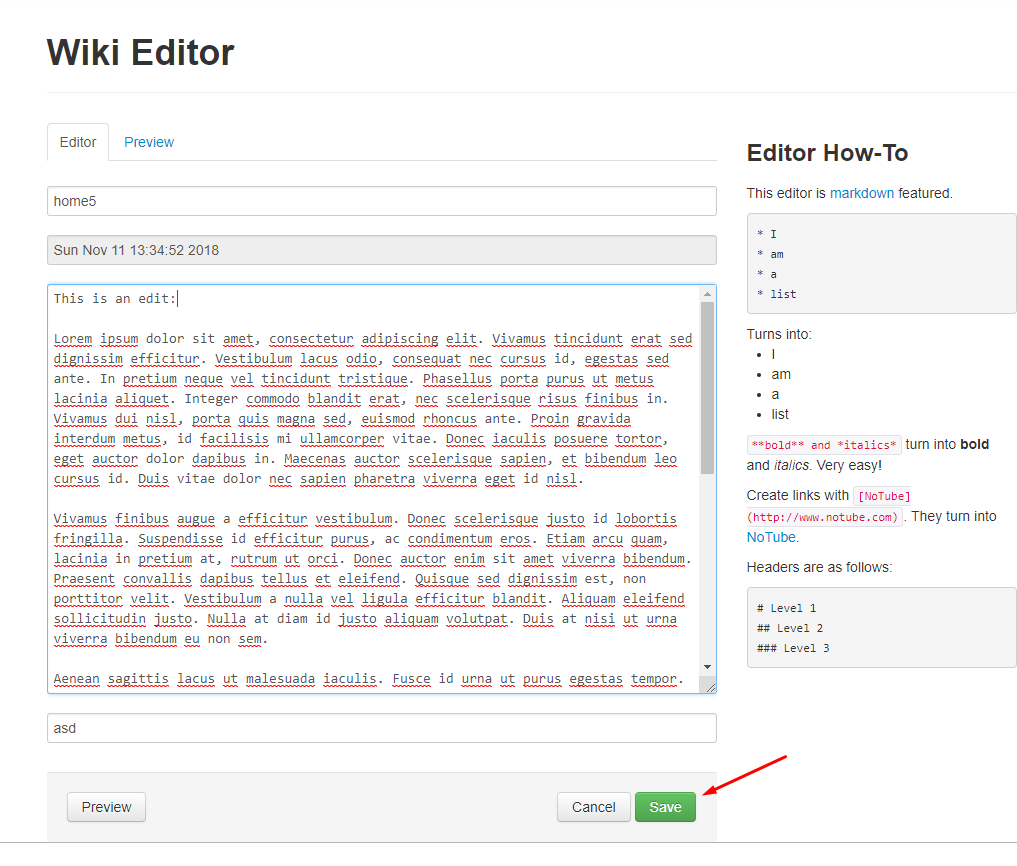
Check if the file and directory were made, and see if the date was changed from the original.

# User manual

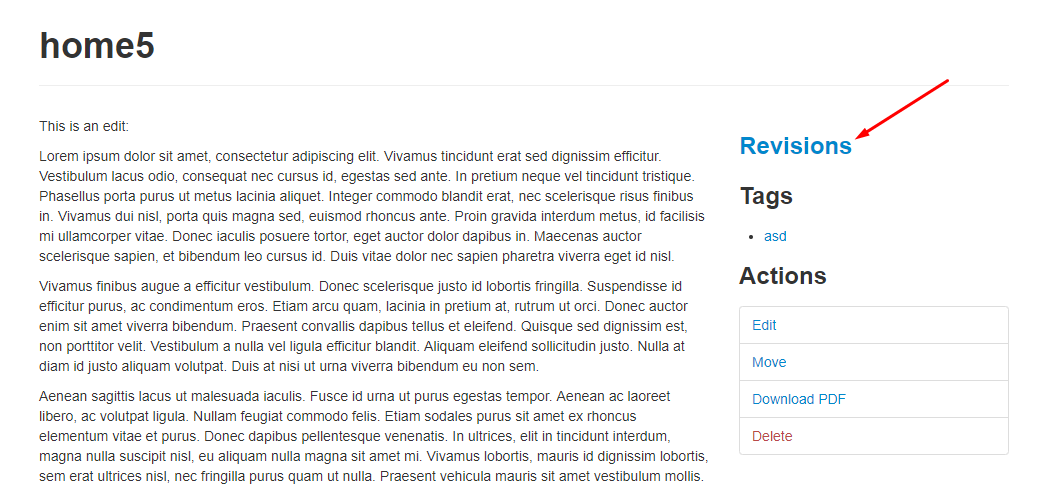
Using this feature is simple; you will be able to edit without worry; allow other users to edit without missing out on any information that may have been edited out; and you will also be able to see when pages are edited.

Here is a simple guide to get you started

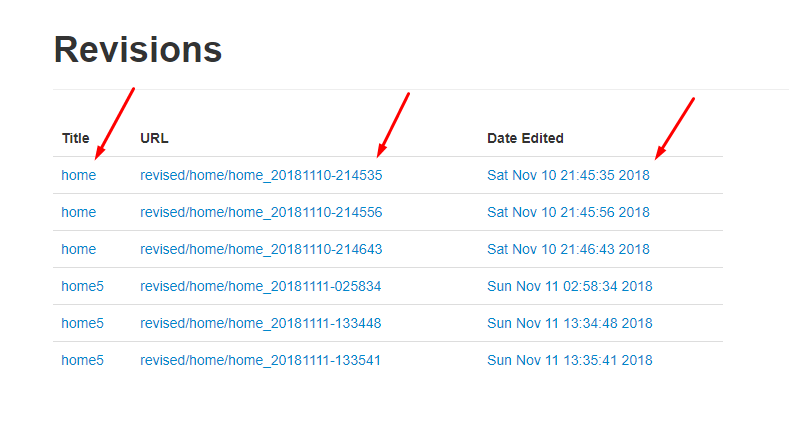
## Step 0: Edit an article or have someone else edit an article



## Step 1: Click on the “Revisions” button on the sidebar on a specific page

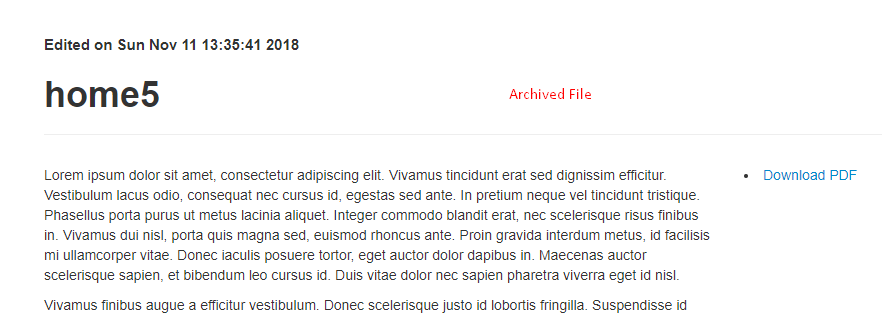
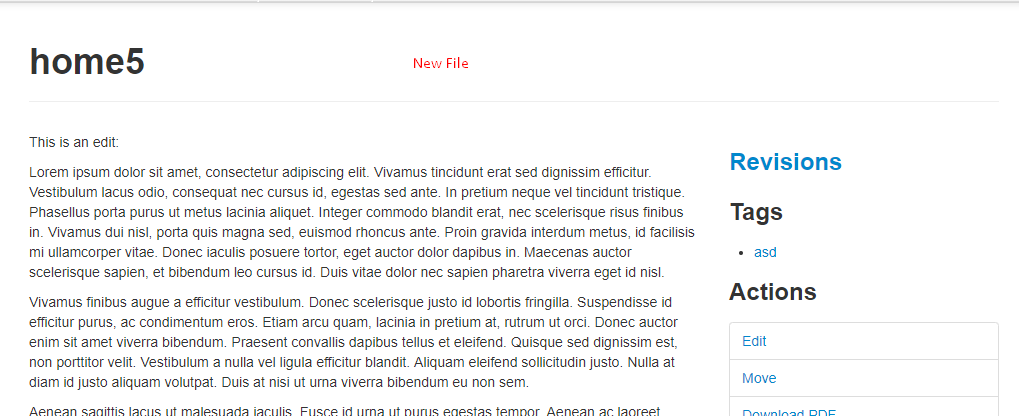


## Step 2: Select the revision you would like to view



## Step 3: View (and compare)

You can also Download the PDFs of Previous versions of the page!



# Testing documentation

Testing for this feature was fairly simple, we first test our save() to make sure my edit did not mess anything up, and then we test the archive() method. The easiest way to test that our archive() method works as intended, is to see if the folder and file were created and that they weren’t there already: to achieve this, the following unittests were made

def test\_page\_saving(self):  
 *"""  
 Assert that saving a page back to disk persists it  
 correctly.  
 """* self.page.save()  
 with open(self.page\_path, 'r', encoding='utf-8') as fhd:  
 saved = fhd.read()  
 assert saved == self.page\_content  
  
def test\_page\_archive(self):  
 *"""  
 Checks the before and after archive, Making sure the /revised/ folder and   
 file are created, but did not exist before  
 """* self.assertEqual(os.path.exists(self.page\_relative + "/revised/"), False)  
 self.page.archive(self.page\_relative)  
 assert os.path.exists(self.page\_relative + "/revised/")  
 self.assertEqual(os.listdir(self.page\_relative + "/revised/"), ['test'])

Jacob deserves a lot of the credit for making this as simple as it is, since he built the basic test environment.