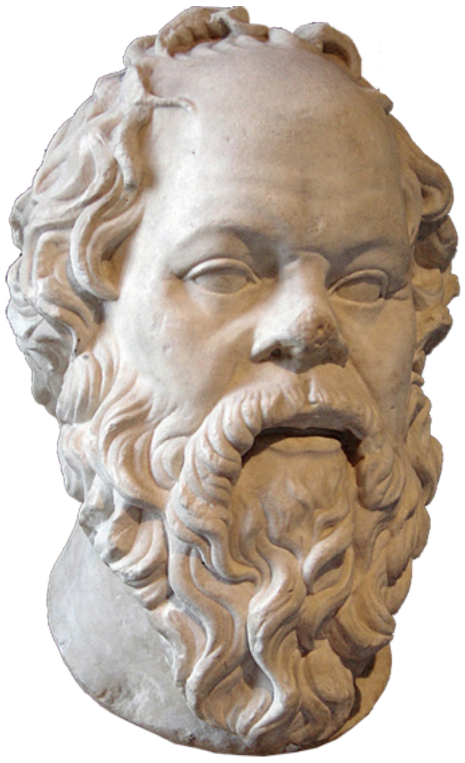

writing an RSS reader in 80 lines of bash

I consume most of my information through RSS. RSS is amazing. Up until yesterday, I used Newsboat to go through my feed, and had a `,a` macro that appended a link from the item to a text file with links. Every morning, I would go through my whole feed, and get a bunch of links appended to my `links.md` file. After that, I start reading them by copying one line at a time, and running the script that dumps the content of the link via w3m and opens a vim buffer with it. If it is a large read, I would save it as a txt file and send it to my e-reader.

Recently, I started reducing my dependencies on external software. Yesterday, I thought, why not write an RSS reader myself. It will not be as powerful and robust as my current one, but it will be simpler, more hackable, and mine, i.e. I will not be dependent on a stranger's commit that might break my system accidentally or on purpose.

First, I wanted to write it in Zig, as this is the language I have been recently playing with and want to get better in. But then I thought that omnipresent Unix utils like `curl`, `grep`, `awk`, `sed` will be enough to quickly hack a simple script. I had the following idea. Write a bash script that will `curl` a given feed, extract the title/link/description from each item, apply filters on each of the fields above (e.g. I do not want to get hackernews posts on TypeScript), and append the results to a file. Later, I will run this script with `xargs` in parallel to quickly get all the feeds downloaded.

This is, basically, how it works:

```
cat $HOME/notes/urls.txt | xargs --verbose -P $THREADS -I {} bash -c
'~/dev/yr/yr_feed "{}" | flock -x $HOME/dev/yr/new.txt -c "cat >>
$HOME/dev/yr/new.txt"'
```

I added `flock` on the file append at the end to make sure that the writes from multiple threads do not get interleaved.

Let's see what do I do with each of the feeds now!

```
url="$1"
mapfile -t items < <(curl -s "$url" | awk 'BEGIN{RS="<(item|entry)>"; ORS=""}
NR>0 {
    gsub(/\n/," ");       # replace newlines with spaces
    gsub(/[[:space:]]+/," "); # collapse multiple spaces
    print $0 RS "\n"
}' | head -n $MAX_ITEMS)
```

Here, we download the feed, and split the received xml by item/entry tags using `awk`. We limit the number of items to process (often the feed items are just appended to the file, and we do not want to go over a 1000-item list every time). We also remove newline symbols within each item as I will use `grep` to parse elements of items later.

We are now ready for a for loop over the items array, where we parse the content and decide whether to append it to our reading list.

```
link1=$(echo "$item" | grep -oP ']*>\K.*?(?=)')
link2=$(echo "$item" | grep -oP ']*\bhref="\K[^"]+')
title=$(echo "$item" | grep -oP ']*>\K.*?(?=)')
hash=$(echo -n "${link1}${link2}${title}" | md5sum | awk '{print $1}')
```

We first find the two main elements of the feed item: its title and its url. I get `link1` and `link2` because some of the feeds have their urls using a style, potentially with some attributes inside. Others use `<link href="">` style, also potentially with attributes. Title is more or less standard. I will probably benefit from reading the RSS format specs, but I did not yet have time to look those up.

After that, I use those to compute a hash of an item. We need this hash to understand whether we have already downloaded the item or not. The most important thing here is not to use content to compute the hash. Some of the websites, e.g. hackernews, put the number of comments on a post into the content. This makes the same thing appear in my reading list multiple times. We do not want that. This potentially misses updates for pages that add content gradually. I will need to deal with this somehow in the future iteration of the script.

```
if ! grep -Fxq "$hash" "$DOWNLOADED_PATH"; then
    content1=$(echo "$item" | grep -oP ']*>\K.*?(?=)' | cut -c1-$MAX_CHARS |
w3m -T text/html -dump)
    content2=$(echo "$item" | grep -oP ']*>\K.*?(?=)' | cut -c1-$MAX_CHARS |
w3m -T text/html -dump)
    [[ -n $link1 ]] && [[ $(match_against_filters "$link1" "url") -eq 1 ]] &&
continue
    [[ -n $link2 ]] && [[ $(match_against_filters "$link2" "url") -eq 1 ]] &&
continue
```

```
    [[ -n $title ]] && [[ $(match_against_filters "$title" "title") -eq 1 ]]
&& continue
    [[ -n $content1 ]] && [[ $(match_against_filters "$content1" "content") -
eq 1 ]] && continue
    [[ -n $content2 ]] && [[ $(match_against_filters "$content2" "content") -
eq 1 ]] && continue

    echo "--- $1"
    [[ -n $link1 ]] && echo "$link1"
    [[ -n $link2 ]] && echo "$link2"
    [[ -n $title ]] && echo "$title"
    [[ -n $content1 ]] && echo "$content1"
    [[ -n $content2 ]] && echo "$content2"
    echo

    new_hashes+=$'\n'"$hash"

fi;
```

We are now ready to parse the content and apply our filters! I have been a heavy user of filters in Newsboat, carefully ~~building my own information bubble~~ protecting myself from the information overdose. Again, content can be a description/summary tag, and we check both. After that, we check if any of the fields match the list of filters we have in a separate file. If they do, we go to the next item on the list, otherwise, we print out the item title/url/content and append the item to the list of hashes we do not want to download again.

```


match_against_filters() {
    local link="$1"
    local type="$2"

    while IFS=',' read -r feed type regex; do
    if ! echo "$link" | grep -Eq "$regex"; then
        return 1
    fi
    done < <(grep "^\*,${type}" "$FILTERS_PATH")

    while IFS=',' read -r feed type regex; do
    if ! echo "$link" | grep -Eq "$regex"; then
        return 1
    fi
    done < <(grep "${url},${type}" "$FILTERS_PATH")

    return 0
}
```

The matching is pretty simple. We can have three fields to match (url, title, and content), and two rules to define whether we want to match a line or not. The * symbol means that a filter should be applied to every item. E.g, I have one that filters out all medium/substack links `*,url,"medium|substack"`. Another one blocks youtube shorts videos: `*,url,".*youtube.com/.*shorts"` If we use a feed url instead of the star symbol, the filter will only be applied to a particular feed. I use this primarily to filter out a list of `arxiv.org` submissions I go over every morning.

Short disclaimer before we continue: I haven't thorougly tested the filter system yet, there are possible bugs in there. Work in progress, tbd.

Finally, when we have processed all the items, we append the list of downloaded hashes to the `downloaded.txt` file. Again, there is a file lock on the append operation as this script will be used in parallel for multiple feeds.

```
{
  flock 9
  printf "%s" "$new_hashes" >&9
} 9>>"$DOWNLOADED_PATH"
```

That's it, really! You can have a simple RSS reader only depending on core Unix utils that are everywhere. Sometimes, the output will be ugly, it will not work for some of the feeds due to naive parsing. But it can be improved, it was fun to build, and it is pretty usable. Hope it was fun. Here is the full code in case you want to play with it.

```
#!/bin/bash

MAX_ITEMS=150
MAX_CHARS=1000
DOWNLOADED_PATH="$HOME/dev/yr/downloaded.txt"
FILTERS_PATH="$HOME/dev/yr/filters.txt"
touch "$DOWNLOADED_PATH"

# filters syntax: URL,type,REGEX
# if * is used for URL, then it is applied to all urls.
# types: title/url/content
# grep will do negative match! E.g. if I hit a find, I will skip an item.

new_hashes=""

match_against_filters() {
    local link="$1"
    local type="$2"

    while IFS=',' read -r feed type regex; do
    if ! echo "$link" | grep -Eq "$regex"; then
        return 1
    fi
    done < <(grep "^\*,${type}" "$FILTERS_PATH")

    while IFS=',' read -r feed type regex; do
    if ! echo "$link" | grep -Eq "$regex"; then
        return 1
    fi
    done < <(grep "${url},${type}" "$FILTERS_PATH")

    return 0
}

url="$1"
```

```
mapfile -t items < <(curl -s "$url" | awk 'BEGIN{RS="</(item|entry)>"; ORS=""}
NR>0 {
    gsub(/\n/," ");       # replace newlines with spaces
    gsub(/[[:space:]]+/," "); # collapse multiple spaces
    print $0 RS "\n"
}' | head -n $MAX_ITEMS)

for item in "${items[@]}"; do
    link1=$(echo "$item" | grep -oP ']*>\K.*?(?=)')
    link2=$(echo "$item" | grep -oP ']*\bhref="\K[^"]+')
    title=$(echo "$item" | grep -oP ']*>\K.*?(?=)')

    hash=$(echo -n "${link1}${link2}${title}" | md5sum | awk '{print $1}')
    if ! grep -Fxq "$hash" "$DOWNLOADED_PATH"; then

        content1=$(echo "$item" | grep -oP ']*>\K.*?(?=)' | cut -c1-$MAX_CHARS
| w3m -T text/html -dump)
        content2=$(echo "$item" | grep -oP ']*>\K.*?(?=)' | cut -c1-$MAX_CHARS
| w3m -T text/html -dump)
        [[ -n $link1 ]] && [[ $(match_against_filters "$link1" "url") -eq 1 ]]
&& continue
        [[ -n $link2 ]] && [[ $(match_against_filters "$link2" "url") -eq 1 ]]
&& continue
        [[ -n $title ]] && [[ $(match_against_filters "$title" "title") -eq 1
]] && continue
        [[ -n $content1 ]] && [[ $(match_against_filters "$content1"
"content") -eq 1 ]] && continue
        [[ -n $content2 ]] && [[ $(match_against_filters "$content2"
"content") -eq 1 ]] && continue

        echo "--- $1"
        [[ -n $link1 ]] && echo "$link1"
        [[ -n $link2 ]] && echo "$link2"
        [[ -n $title ]] && echo "$title"
        [[ -n $content1 ]] && echo "$content1"
        [[ -n $content2 ]] && echo "$content2"
        echo

        new_hashes+=$'\n'"$hash"

    fi;
done

# TODO think on how to clean that file later
{
  flock 9
  printf "%s" "$new_hashes" >&9
} 9>>"$DOWNLOADED_PATH"
```