bioRxiv preprint doi: https://doi.org/10.1101/2020.08.03.232843; this version posted August 5, 2020. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

# Bashing irreproducibility with *shournal*

Tycho Kirchner[1], Konstantin Riege[1], Steve Hoffmann*[1]

[1]Computational Biology Group, Leibniz Institute on Aging — Fritz Lipmann Institute (FLI), Jena, 07745, Germany

Received on XXXXX; accepted on XXXXX

*Corresponding Author: steve.hoffmann@leibniz-fli.de

bioRxiv preprint doi: https://doi.org/10.1101/2020.08.03.232843; this version posted August 5, 2020. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.


## Abstract

Arguably, one of the most important tools for computer science is the Linux shell. Processing steps carried out there are critical for many analyses and software development projects. However, manual documentation of the work is time-consuming and error-prone. To remedy this problem, *shournal* tightly integrates with the shell and automatically records all shell commands along with associated file events. For any file, *shournal* allows the reconstruction of the command history and is able to create detailed reports for whole project directories. *shournal* is based on the fanotify API and mount namespaces and allows the efficient monitoring of entire process trees.

**Availability:** The code for *shournal* is freely available at https://github.com/tycho-kirchner/shournal under the GNU General Public License v3.0 or later

**Contact:** steve.hoffmann@leibniz-fli.de

# 1 Introduction

The daily work of many computer scientists involves the Linux shell. Despite its limited graphic capabilities and the complex syntax, its unmatched flexibility makes it the tool of choice for many file operations such as sorting or concatenation as well as for writing small scripts and pipelines. Especially the *pipe*, allowing the user to combine various Linux tools and scripts into a single pipeline substantially increases the shell's usability. However, in order to keep track of the work process, e.g., in the context of larger analysis projects, it quickly becomes necessary to maintain README files with great diligence to ensure reproducibility. In cases where such documentation is incomplete or not available, the shell's history might be used to manually reconstruct the chain of commands that have been used to create and modify individual files. However, this process is rather time-consuming, error-prone, and leaves some critical blind spots. A reconstruction of the work can quickly become impossible if the programs that were used to create or modify the files in question are not available anymore or have been changed without proper version control. For smaller ad-hoc shell, awk, perl, or python scripts such version control is often omitted, inherently posing a threat to the reproducibility of the work. Several workflow engines such as *Snakemake* or *Nextflow* (Köster and Rahmann, 2012; Di Tommaso, 2017) have been proposed to alleviate this vital problem for the scientific process. For practical or motivational reasons, however, analysis projects are not always directly initiated or consequently carried out within these frameworks. Thus, building such a workflow just prior to data publication can quickly become a nightmare. To reduce this blind spot and to remedy some of the shortcomings of the shell, we have developed *shournal*, a Linux program that makes use of the Linux kernel's *fanotify* module to document and summarize the work process for user-defined project paths.

bioRxiv preprint doi: https://doi.org/10.1101/2020.08.03.232843; this version posted August 5, 2020. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

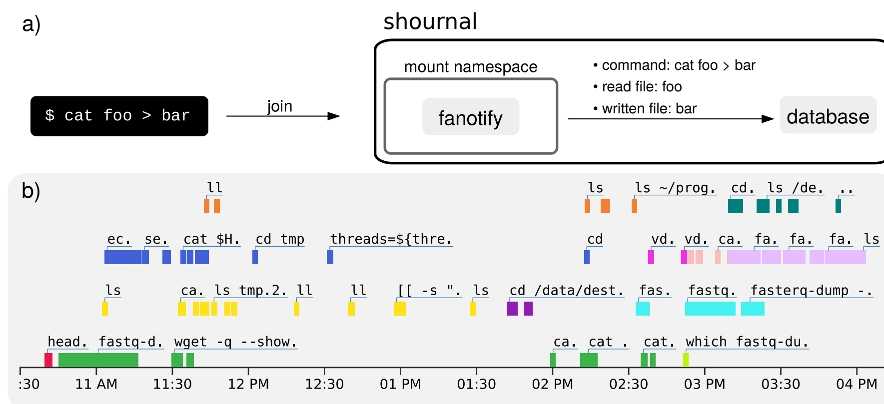


Figure 1: *shournal's* architecture and output options. (a) Schematic illustration of a shell-session observed by shournal. The concerned processes of the shell-command `cat foo > bar` join a unique *mount namespace* where file operations are observed by *fanotify*. Metadata and checksums of the read file (*foo*), as well as the written file (*bar*), are stored in the database alongside the corresponding shell-command. (b) Visualization of the shell command history. *shournal* can export the command history into an interactive Html-plot. Commands, which were executed within a given shell session, are marked with the same color, parallel shell sessions are vertically stacked.

bioRxiv preprint doi: https://doi.org/10.1101/2020.08.03.232843; this version posted August 5, 2020. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

# 2 Materials and Methods

## 2.1 Logging the command history

*shournal* deterministically tracks file events by using low-level operating-system capabilities of the Linux-kernel. Specifically, file access events can be exposed to the *user-space* by *fanotify*. The *fanotify* API allows subscribing for file-events of a specific mount-point. The *unshare* system call detaches a process from its parent *mount namespace*. Since *mount namespaces* are inherited by new processes and can be joined by other processes, *fanotify* can be used to subscribe to file-events of a group of processes. Using *shournal's* shell-integration, realized via a *shared library* which can be loaded into a shell's process, it is ensured that all processes created during the execution of a command join the same *mount namespace*, which is observed by *fanotify* (Fig. 1a). Similarly, also file redirections occurring within the shell-process refer to the new *mount namespace*. This approach enables *shournal* to systematically track reading and writing events that were triggered by the shell or any of its spawned processes. Therefore, journaling includes all such events irrespective of whether they are triggered by shell commands, programmes with graphical user interfaces (GUIs), or scripts. In the latter case, controlled by user-defined size restrictions and file extensions such as *.sh* or *.pl*, *shournal* also archives the entire script. This facilitates the reproduction of results obtained from smaller scripts where a version control was initially deemed unnecessary.

## 2.2 Querying and visualizing the command history

*shournal* allows flexible queries by using a *sqlite* database to store the data about file operations as well as the scripts. For instance, for a given file, it can be queried what shell-command created or modified it. Other options include the query for individual files modified during a given period, the command-history at a designated project directory, or commands executed during a specific shell session. To make the command history more accessible and in addition to an output on the console, *shournal* generates an interactive graphical map of commands for user-specified files, directoriesi, and/or dates (Fig. 1b). The map displays each shell session in an individual row and thus allows to better identify specific chains of subsequently executed commands. Clicking on a command displayed in the interactive map gives supplementary information on the exact time of execution, archived scripts, or checksums. Further miscellaneous statistics are displayed in bar-plots, e.g., the commands with most file-modifications. The collected data can be shared with other programs using the JSON output format. Due to the low-level nature of the data, it can be used as a basis for higher-level systems such as workflow managers. For instance, an observed shell-command-series can be directly transformed into rules for the *Snakemake workflow engine* (Köster and Rahmann, 2012) using the software at https://github.com/snakemake/shournal-to-snakemake. The input- and output-section of a rule is generated from the captured file-events.

bioRxiv preprint doi: https://doi.org/10.1101/2020.08.03.232843; this version posted August 5, 2020. The copyright holder for this preprint (which was not certified by peer review) is the author/funder. All rights reserved. No reuse allowed without permission.

# 3 Requirements

*shournal* depends on the Linux kernel's *fanotify* and *mount namespaces*; thus, it runs on Linux systems only. For Debian- and Ubuntu-based distributions, deb-packages are available on the release-page. In order to unshare or join *mount namespaces* and to initialize *fanotify*, *shournal* needs to be run as a *set-user-ID* (setuid) program (Kerrisk, 2010, chap. 9.3). However, setuid privileges are only used during startup. All other operations are done with effective user rights. Further information is provided on *shournal's* github page.

# 4 Conclusion

*shournal* allows to comprehensively record, search and visualize the work carried out on the Linux shell. Therefore, it can play an essential role in facilitating the reproducibility of work carried out on the linux shell, e.g. during bioinformatic analyses, by allowing to summarize, review, and to resume older projects quickly. By implementing the *shournal-to-snakemake* converter, we demonstrate that the *shournal* output may be used as a basis for the creation of pipelines with *Snakemake*. In principle, similar converters may be written for other workflow managers, e.g., via the JSON or YAML-based Common Workflow Language (CWL) (Amstutz, 2016). Alternatively, the CWL-described workflow may be run directly within other managing systems such as Nextflow (Di Tommaso, 2017). By narrowing the gap between the often poorly documented ad-hoc work on the shell and the generation of sophisticated workflows, *shournal* can help to improve the scientific practice.

# Funding

This work has been supported by the BMBF project de.STAIR (031L0106D).

# References

Di Tommaso, P. et al. (2017) Nextflow enables reproducible computational workflows. *Nature Biotech.*, **35**, 316319

Kerrisk, M. (2010) The Linux Programming Interface. No Starch Press, Inc., San Francisco.

Köster, J., and Rahmann, S. (2012) Snakemakea scalable bioinformatics workflow engine. *Bioinformatics*, **28**, 25202522.

Amstutz, P. et al. (2016): Common Workflow Language, v1.0. Specification, Common Workflow Language working group. *https:/w3id.org/cwl/v1.0/ doi:10.6084/m9.figshare.3115156.v2*

a)

shournal

$ cat foo > bar

join

mount namespace

fanotify

- command: cat foo > bar
- read file: foo
- written file: bar

database

b)

ll

ls

ls ~/prog.

cd.

ls /de.

..

ec.

se.

cat $H.

cd tmp

threads=${thre.

cd

vd.

vd.

ca.

fa.

fa.

fa.

ls

ls

ca.

ls tmp.2.

ll

ll

[[ -s ".

ls

cd /data/dest.

fas.

fastq.

fasterq-dump -.

head.

fastq-d.

wget -q --show.

ca.

cat .

cat.

which fastq-du.

:30

11 AM

11:30

12 PM

12:30

01 PM

01:30

02 PM

02:30

03 PM

03:30

04 PM