Cuadernos de
Teoría Social
Año 4, Nº 7

ARCHIVOS DE CAMBIO
Los grandes repositorios de software como infraestructuras digitales volátiles

Maxigas
Lancaster University, Reino Unido
| maxigas@lancaster.ac.uk |

Resumen
Los repositorios archivan cambios puntuales en el código de software antes que en documentos originales en lengua natural. El repositorio es una forma diagramática que posibilita el ensamblaje de cambios en intrincados patrones temporales. Mientras que los repositorios *Git* expresan la historia particular de un proyecto de software en tanto estructura árbol, con ramas que se bifurcan y combinan, los repositorios *Debian* presentan la liberación de colecciones de software acumuladas en capas geológicas, una sobre la otra. Dada la volatilidad del software y la naturaleza abierta de dichas temporalidades, quienes mantienen estos repositorios deben repararlos constantemente para contener versiones actualizadas del software auténtico. Por ello, las preocupaciones centrales de los archivistas y el rol del trabajo humano son más importantes que nunca para la viabilidad de infraestructuras digitales que están a la base de una vida cotidiana mediada tecnológicamente. Las inscripciones técnicas a través de automatización no suprimen, sino que ponen en primer plano el trabajo humano requerido para mantener los presupuestos de archivo que garantizan la continuidad de la civilización y cultura modernas.

Palabras Claves
Archivo, Infraestructura, Software, Temporalidad, Mantención

Abstract
Repositories archive punctual changes in software code rather than original natural language documents. The repository is a diagrammatic form that defines changes to be assembled into intricate temporal patterns. While Git repositories express the history of a particular software project as a growing tree structure with branches that fork and merge, Debian repositories present releases of software collections that accumulate as the layers of geological strata on the top of one another. Due to the volatility of software and the open-ended nature of such temporalities, maintainers continuously repair repositories to contain up-to-date, authentic versions of software. Therefore, the central concerns of

archivists and the role of human labour are more important than ever for the viability of digital infrastructures that ground technologically mediated everyday life. Technical inscription through digital automation does not abolish but rather foreground the human labour required to maintain the archival presuppositions that ensure the continuity of modern civilisation and culture.

Keywords
Archive, Infrastructure, Software, Temporality, Maintenance

Introducción

La transitoriedad de los nuevos archivos, su siempre corta media-vida, es su destino, su maldición, y su oportunidad
– Wolfgang Ernst (2005: 99)

Teorizando acerca de los archivos, Velody ha escrito que "las apelaciones a la verdad última, junto a la adecuación y plausibilidad del trabajo de las humanidades y las ciencias sociales descansa en presupuestos de archivo" (Velody, 1998: 1). Varios autores, como Featherstone (2006), Ernst (2013) y Chun (2011), expresan ansiedad respecto a la mantención continua y la viabilidad de los presupuestos de archivo de la era digital. La falta de cuidado por estas cuestiones parece amenazar el proyecto de la modernidad.

En este breve texto muestro cómo los presupuestos de archivo sustentan populares infraestructuras digitales contemporáneas tales como los repositorios de software. Los repositorios de software presentan programas en una forma estructurada para el día a día de usuarios de sistemas operativos, desarrolladores de software y administradores de sistemas. Mientras que los programas individuales pueden ser instalados, distribuidos, inspeccionados, modificados y usados sin tener que acudir a repositorios, estos últimos ayudan a los usuarios a realizar estas tareas. De

hecho, en estos momentos buena parte del software que es instalado, modificado y usado en el mundo pasa a través de un repositorio u otro.

Quienes mantienen grandes repositorios de software se ven enfrentados a preguntas clásicas respecto a los archivos, especialmente en lo referido a su dimensión temporal, pero en un contexto sociotécnico bastante alejado del trabajo de archivo clásico. Si bien los mantenedores buscan soluciones viables en términos de autenticidad, adecuación y plausibilidad, ellos también transfiguran de maneras interesantes aquello que los archivos pueden llegar a ser. Aquí deseo teorizar el giro desde los archivos a los repositorios en términos de la problemática de archivar los cambios [*archiving changes*].

El punto de partida de mi argumento es el adagio hacker que plantea que "el código es discurso", de donde se sigue el corolario de que el código de software merece el mismo respeto que otras formas de producción, desempeño y expresión lingüística. Esto resuena con el programa de investigación de los *Software Studies* en tanto examina el código fuente como parte integrante de la cultura material. Es decir, como un factor significativo que estructura la vida social contemporánea en la forma de infraestructuras digitales (ver, por ejemplo, Fuller, 2003). Los materiales empíricos a los que referiré para respaldar mis afirmaciones se restringen a documentación autorizada y públicamente accesible respecto a la arquitectura técnica, convenciones sociales y uso real de las infraestructuras digitales. Sin embargo, mi análisis está también informado por mi experiencia en la industria como desarrollador de software, administrador de sistemas y usuario de software libre, además de mi trabajo de campo en curso, iniciado en 2012, sobre las infraestructuras digital, material y temporal del *hacking* como una cultura ingenieril alternativa (ver, Maxigas 2012; Maxigas 2017).

Los dos casos empíricos que mencionaré son los "repositorios *Git*" –el formato más popular de almacenamiento, desarrollo y distribución de código fuente hoy en día–, y el "repositorio *Debian*" –el almacenamiento

de software de un sistema operativo icónico de software libre que contiene versiones binarias ejecutables listas para ser instaladas. Los repositorios *Git* individuales contienen el código fuente de un solo proyecto de software en tanto conjunto de cambios realizados por desarrolladores en el código base. El repositorio *Debian* contiene versiones estables y actualizadas de aplicaciones preparadas para ser instaladas (tal como *Google Play Store* presenta aplicaciones para ser instaladas en *smartphones* operando con *Android*). Los voluntarios del proyecto *Debian* emplean código fuente desde los repositorios *Git* de proyectos de software individuales y los convierten en paquetes reunidos en el repositorio *Debian* central. Así, éstos median entre los desarrolladores de software y los usuarios finales; preparando la producción de los primeros para que sean útiles para el consumo de estos últimos.

Los repositorios como archivos e infraestructuras

En la mantención y desarrollo de software, un repositorio se comprende como una colección de código de software. Los repositorios se ubican entre los archivos y las bases de datos. Tal como un archivo, un repositorio es producto del ejercicio curatorial humano que se ciñe a un conjunto de criterios; asimismo, quienes lo mantienen actúan como sus guardianes frente al público. Tal como una base de datos, un repositorio es almacenado y accedido en un formato digital que permite la búsqueda y recuperación eficiente de información, incluso si es que sus contenidos cambian rápidamente. Finalmente, un repositorio está primordialmente estructurado en dimensiones temporales: actualizaciones graduales que forman capas, árboles e historias intrincadas donde la autenticidad es asegurada por un conjunto de procesos automatizados y decisiones humanas.

El *Git* es un "sistema de control de versiones distribuido" para desarrolladores de software[1]. En otras palabras, almacena cambios en el código fuente[2]. Los repositorios *Git* revolucionaron las prácticas de desarrollo de software al permitir la cooperación flexible entre un gran número de colaboradores para proyectos complejos sin perder de vista las distintas versiones del código base (Dafermos, 2012). Cada cambio es firmado por un autor, lleva un sello de tiempo y un mensaje de *commit* en una lengua natural que describe el propósito y efecto del cambio, tales como agregar una nueva característica o arreglar un error[3].

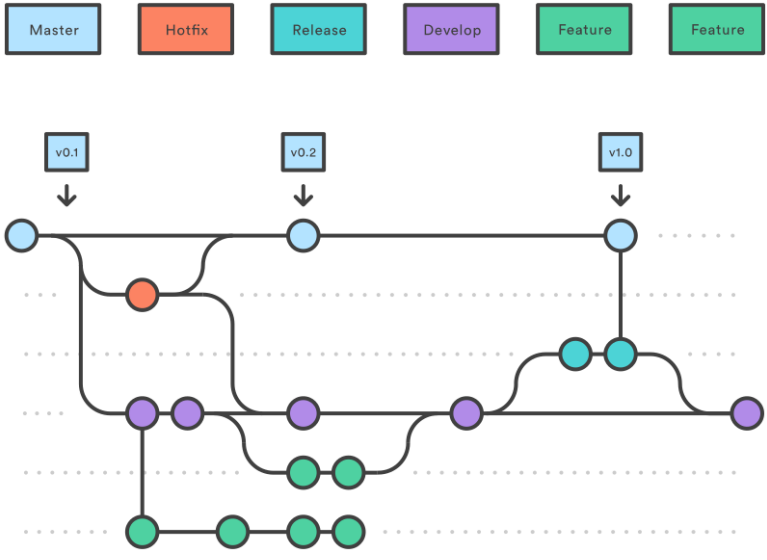

Figura 1. Diagrama de flujo de trabajo "Gitflow" del Tutorial *Git*.
Fuente: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

[1] Ver, web oficial de Git: https://git-scm.com/
[2] Ver, documentación oficial de Git: https://git-scm.com/docs/gittutorial
[3] Ver, documentación oficial de Git: https://git-scm.com/docs/git-commit

Estos cambios forman una estructura de árbol la cuál es conocida como la *historia* del software (figura 1). Un repositorio *Git* bien mantenido cuenta una historia de desarrollo del software de manera tal que responde a las demandas de archivo respecto de decir la verdad en relación con la historia, incluyendo trayectorias de desarrollo paralelo, cabos sueltos y versiones combinadas[4]. Existen procedimientos específicos para *reescribir la historia* en *Git*, pero son técnicamente engorrosos y culturalmente rechazados como malas prácticas[5]. Las prácticas de desarrollo contemporáneo reconocen el rol de los maestros de la fusión [*merge masters*], quienes aseguran que se sigan los procedimientos adecuados cuando se están agregando cambios en el repositorio. Así, son responsables por la verdad, adecuación y plausibilidad de un repositorio particular[6].

El proyecto *Debian* es un sistema operativo de software libre emblemático que puede ser instalado en computadores personales[7]. El proyecto es respaldado por una fundación, estructuras burocrático-meritocráticas y un fuerte ethos hacker (Coleman, 2012). Documentos fundacionales tales como el "Debian Social Contract" y las "Free Software Guidelines" gobiernan la mantención y desarrollo de *paquetes* Debian[8].

De hecho, *Debian* fue pionero, en 1998, en la *administración de empaquetado* [*package management*] para la distribución de software[9]. Por

[4] Entrevista a *efkin*, desarrollador senior de una empresa mediana, 18 de Agosto de 2018.

[5] "Rebasar (o cualquier otra forma de re-escribir) una rama en la que otros han situado su trabajo es una mala idea: *quienquiera que esté en esta corriente está forzado a arreglar manualmente su historia*" (el énfasis es mío). Documentación oficial de Git: https://git-scm.com/docs/git-rebase

[6] Ver, en la plataforma de social media Reddit, la pregunta "¿Cómo ser un buen merge master?". La primera respuesta empieza con un "Nunca jamás re-escribas la historia…".
https://www.reddit.com/r/git/comments/660ohx/how_to_be_a_good_merge_master/

[7] Debian: El Sistema operativo universal. Ver, la página web oficial del proyecto: https://www.debian.org/

[8] Ver, "Debian Social Contract": https://www.debian.org/social_contract

[9] Al respecto, ver "Brief history of Debian, Chapter 4: A detailed history": https://www.debian.org/doc/manuals/project-history/ch-detailed.en.html. El año 2017, Jeremy Katz escribió: "Solamente con el lanzamiento de la apt-get de *Debian* en 1998 y la up2date de Red Hat en 1999 tú puedes empezar a bajar e instalar fácilmente los paquetes, con todas sus dependencias, y sin tener que explicitarlas todas": https://blog.tidelift.com/a-brief-history-of-package-management. También en 2017, Jan Krutisch añade: "Unos pocos años después las distribuciones de Linux popularizaron la idea de 'package managers', siendo la idea el que tu

ese entonces había muy pocos sistemas operativos de software libre que ofreciesen un mecanismo integrado para decenas de miles de paquetes de software funcionando juntos. Pero lentamente la idea se volvió popular. Desde entonces, esta aproximación ha sido adoptada por grandes corporaciones en la forma de servicios de distribución tales como la "Mac App Store" para dispositivos Apple (2007), "Google Play" para smartphones con Android (2008), o el "Microsoft Store" para plataformas Windows (2012).

Todas estas plataformas digitales de distribución son presentadas a los usuarios como bases de datos que admiten búsquedas y donde programas individuales pueden ser instalados/desinstalados del computador[10]. Los desarrolladores pagan para incluir su software en las *app stores* corporativas, y los usuario usualmente pagan también para instalar ciertas apps. De esta manera, el archivo central de integración de sistemas se convierte en un elemento clave para el régimen de acumulación de capital contemporáneo: un instrumento de búsqueda de rentas lucrativas. En contraste, los paquetes *Debian* son siempre libres de cargo y la labor de empaquetado no es remunerada por el proyecto (no obstante, algunas compañías tienen empleados que hacen empaquetado *Debian*). Dado que la administración de paquetes se considera una buena práctica ingenieril, *Debian* jamás ha reclamado propiedad sobre la idea o disputado otras implementaciones. Los desarrolladores de software libre generalmente toman la postura de no rivalizar con sus competidores.

Cada dos años *Debian* publica una nueva versión *estable* del sistema operativo, la cual se compone por más de 50,000 paquetes[11].

pudieses instalar los programas y sus dependencias sin tener que recopilarlos todos de manera manual. Ejemplos populares son dpkg/apt de Debian/Ubuntu y RPM de Red Hat. Si bien existen diferencias entre estos administradores de paquetes a nivel de sistema y aquellos a nivel de lenguaje, ambos comparten varios conceptos. Antes que nada, ambos requieren de repositorios de paquetes o, por lo menos, índices de paquetes que apuntan a un lugar donde poder obtener el paquete": https://depfu.com/blog/2017/03/22/a-brief-history-of-dependency-management

[10] Ver, "App Store", en Wikipedia, La Enciclopedia libre: https://en.wikipedia.org/wiki/App_store

[11] Ver, "The Lifecycle of a Release", en The Debian Administrator's Handbook: https://debian-handbook.info/browse/stable/sect.release-lifecycle.html

También están disponibles para los usuarios aventureros las versiones *inestables* y *de prueba* de más o menos los mismos paquetes con nuevas versiones de los mismos programas. En una nueva entrega, los paquetes *inestables* se mueven a la rama *de prueba*, los paquetes *de prueba* se mueven a la *rama estable* y los paquetes *estables* se mueven a la rama de lo *antiguo-estable* [*oldstable*][12]. De este modo, el repositorio *Debian* acumula capas de colecciones de paquetes, con las nuevas versiones del mismo programa empaquetadas en la superficie inestable, bajo la cual existe la posibilidad de recuperar las versiones antiguas de un mismo programa en diversos estratos (figura 2). Los paquetes dejados bajo estas capas son recopilados en el *Archivo Debian*[13]. Los roles de voluntarios en *Debian* incluyen a los *FTP masters*[14]. Este grupo tiene acceso a la escritura del repositorio y en dicha capacidad son, en última instancia, responsables por la verdad, adecuación y plausibilidad del repositorio de software.

[12] Debian Wiki, Releases: https://wiki.debian.org/DebianReleases
[13] README: http://archive.debian.org/README
[14] FTP es la abreviación para "File Transfer Protocol", lo cual se emplea para insertar nuevos paquetes en el repositorio. Ver, Debian Wiki: Teams – FTP Master: https://wiki.debian.org/Teams/FTPMaster

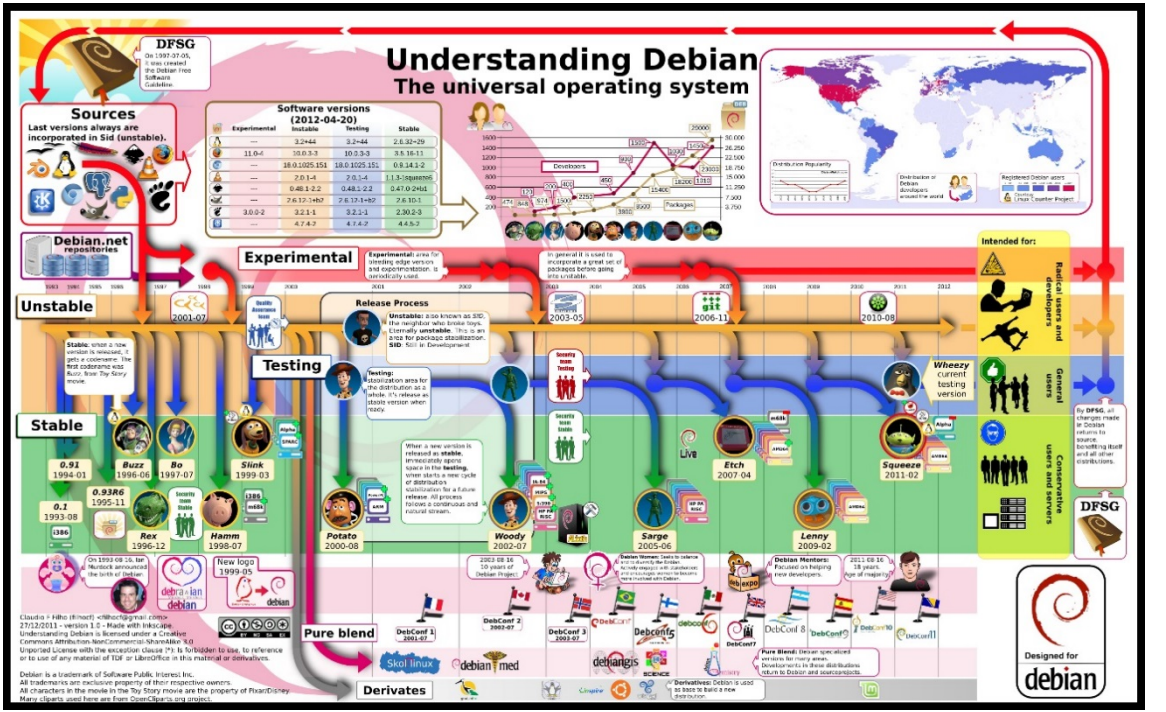

Figura 2. Claudio F. Filho: Poster 'Entendiendo Debian'.
Fuente: https://www.wallpaperflare.com/understanding-debian-wallpaper-181014

Temporalidades diagramáticas

En su investigación sobre la temporalidad ontológica en Deleuze, Kamini (2012) define la temporalidad diagramática como la relación trans-histórica entre pasado, presente y futuro. Esto la relaciona con la noción de diagrama, la cual tiene un rol piloto en tanto "construye lo real que está aún por venir, un nuevo tipo de realidad" (Deleuze y Guattari, 1987: 413). De este modo, "no se sitúa fuera de la historia, sino siempre 'antes de' la historia" (*ibid.*). Recojo aquí esta concepción para teorizar cómo los medios construyen una forma diferente de experimentar el tiempo.

En mis casos empíricos, las preocupaciones de archivo son abordadas a través de una máquina abstracta que produce temporalidades apropiadas para cada infraestructura digital. La diagramática no es inescapable, no obstante, encauza la realidad de una manera cibernética: las temporalidades diagramáticas son máquinas abstractas que gobiernan el tiempo y la historia. Ernst (2016), argumenta que los medios digitales inducen micro-temporalidades; Virilio (2006) advierte acerca de la velocidad de la luz impuesta por las tecnologías digitales en la vida social; Chun (2016) se preocupa por la monotonía inducida por la repetición en las interfaces de los medios sociales contemporáneos. En fin, son incontables las amenazas al proyecto de la modernidad que cuestionan un componente crucial para los archivos: la viabilidad misma de la historia.

En mi teorización acerca de las temporalidades diagramáticas, atiendo a constructos históricos de larga data que los usuarios y las comunidades de desarrolladores inventan para lidiar con las temporalidades inhumanas y autodestructivas de las tecnologías digitales. Irónicamente, hacen esto produciendo aún más software, lo cual lleva a niveles de abstracción más altos. Más que situándose como víctimas de una modernidad tardía que es tan rápida como torpe, tan volátil como cruel, los hackers confrontan estos males de manera material, vale decir, como problemas ingenieriles. Quizás dicha aproximación materialista que busca

la reconciliación con la condición tecnológica contemporánea sea mejor caracterizada en términos menos apocalípticos, esto es, como una modernización reflexiva en línea con los postulados de Beck, Bons y Lau (2003).

Tanto los repositorios *Git* como *Debian* pueden ser comprendidos como sostén de cadenas de eventos organizados de acuerdo con sus propias temporalidades diagramáticas. Las temporalidades diagramáticas están inscritas en el programa *Git* mismo y en la cadena de herramientas de los desarrolladores *Debian*. También se organizan a través de infraestructuras digitales de automatización, aunque quienes trabajan con ellas pueden fácilmente provocar un desastre sin la supervisión de sus *masters*. Por lo tanto, dichas temporalidades no están determinadas tecnológicamente ni son propiedades ontológicas necesarias de los nuevos medios. En días de suerte, ellas son el logro de una cooperación social condicionada por rígidas tradiciones de la cultura ingenieril, así como de comandos inscritos en el software (Akrich, 1992).

Las prácticas de archivo del repositorio *Debian* se organizan a través de la administración del cambio en múltiples escalas temporales que interactúan en ritmos generalmente desalineados. Cada proyecto de software que ha de ser empaquetado sigue su propia trayectoria, incluyendo la primera entrega, periodos de desarrollo más rápidos o lentos, y su eventual obsolescencia. A su vez, el ecosistema como un todo se mueve lentamente en un tiempo glacial que opera de acuerdo con tendencias generales en el desarrollo de software, mientras que los problemas de seguridad son descubiertos y parchados en ondas de choque repentinas. Quienes mantienen el repositorio tienen que asegurarse de entregar una solución de software que sea segura, razonablemente actualizada y, además, bien testeada, para cada una de las tareas mundanas que un usuario pueda querer ejecutar en su computador. Dichos requerimientos corresponden, de modo general, a las preocupaciones por la verdad, la adecuación y la plausibilidad del archivo. Interesantemente,

la preservación en grandes repositorios de software toma la forma de un cambio continuo de contenidos antes que de la conservación de una copia original; no obstante, el propósito de proveer el artefacto auténtico es algo que permanece.

De modo similar, los repositorios *Git* usualmente contienen varias líneas de tiempo interactuando entre sí. Sin embargo, y a diferencia de los repositorios *Debian*, estas líneas temporales están dispuestas en un solo árbol, donde las ramas pueden bifurcarse respecto del tronco y luego, si es que fuese necesario, volver a combinarse con la línea principal. Un caso típico es cuando un desarrollador copia el código fuente actual desde el repositorio y trabaja varios cambios de forma privada mientras implementa una nueva característica o corrige un problema. Luego, los mantenedores del archivo combinan estos cambios con aquellos realizados en el código base, dado que el desarrollador lo copió desde el repositorio original.

Mi punto aquí es que construir estas historias diagramáticas es una cuestión de trabajo humano: una forma de mantención y reparación que gira en torno a las preocupaciones y prácticas clásicas del archivo. En tanto los archivos se mueven desde operar como recursos/fuentes [*(re)sources*] hacia infraestructuras, los cambios producidos en la forma y contenido de los mismos hace que el trabajo de los archivistas sea crecientemente central para su viabilidad. El amplio y profundo despliegue de la automatización inscribe procesos burocráticos en una cinta transportadora digital, la que sin embargo solo funciona si es capaz de absorber la continua entrada de trabajo humano en momentos cruciales para su operación.

Autentificación y autorización

Los problemas de seguridad introducen un vector transversal de cambio para los grandes repositorios de software. Estos problemas normalmente afectan a muchas versiones antiguas y nuevas del mismo software. Cuando

son detectados, dichos problemas son usualmente arreglados tanto para las versiones antiguas como para las más nuevas. Estas intervenciones atraviesan así las temporalidades diagramáticas que definen a los repositorios como infraestructuras, puesto que tanto las versiones estables como inestables del software son actualizadas.

Sin embargo, los repositorios mismos presentan un riesgo para la seguridad. Entendidos como infraestructuras digitales, los grandes repositorios de software son blancos de gran valor. Un vector de ataque común es la falsificación [*forgery*]: por ejemplo, el atacante permuta un paquete en el repositorio por una versión que contiene una vulnerabilidad. Los usuarios que instalen un programa del repositorio serán afectados por la vulnerabilidad. Como consecuencia, quienes mantienen el repositorio deben encontrar formas de asegurarse de que no se produzca una falsificación entre el momento en que pasan sus productos al repositorio y el momento en que los usuarios los retiran. En los términos de la *Seguridad de la Información* [*Information Security*], tales prácticas son tratadas bajo la noción de *autentificación* (Andress, 2014: 40).

Los mantenedores de repositorios emplean variadas estrategias para alcanzar autenticidad en los contenidos. Mientras que los documentos originales en un archivo son objetos relativamente estables, en la medida en que se los deja solos y en condiciones adecuadas, la autenticidad de los grandes repositorios es un logro que depende de ejecuciones virtuosas del trabajo humano. Los *masters* actúan como guardias que brindan acceso a la escritura del repositorio al escoger quienes pueden impulsar nuevos contenidos. Los grandes repositorios de software típicamente poseen sus propios procedimientos burocráticos, los que combinan consideraciones sobre la legitimidad del propósito, contribuciones existentes y reputación establecida. Para el caso del proyecto *Debian*, esto involucra una entrevista tipo examen que considera la motivación, el conocimiento técnico y el compromiso político. El proceso burocrático supone un tipo de trabajo ético a lo largo de una narrativa biográfica-confesional, además de

evaluaciones relativas al conocimiento técnico necesario para el proyecto específico y una familiaridad íntima con los documentos fundacionales de Debian (Coleman, 2012: 147-148).

La mantención del repositorio *Debian* es resultado de la cooperación entre varios equipos, cada uno de los cuales posee varios roles internos. Los mantenedores de *Debian* pueden subir sus propios paquetes al repositorio; por su parte, los desarrolladores también pueden subir paquetes de los contribuidores que patrocinan. Cuarenta y cinco administradores de cuenta están autorizados para recomendar nuevos mantenedores y ayudarlos en la recopilación de la documentación necesaria para aplicar. No obstante, la decisión final depende de tres de los administradores de cuentas. De acuerdo a Steinlin (2009), en 2009 el proyecto contaba con 30,027 contribuidores, pero solamente 2,512 eran mantenedores y 1,330 desarrolladores. Es natural pensar que dichos números hoy son más altos. Pese a lo anterior, son los *FTP Masters* quienes tienen la responsabilidad última por el estado del archivo, incluyendo el procesamiento de nuevas cargas hechas por todos los demás participantes del proyecto[15].

Tras la *identificación* de los participantes legítimos, estos deben ser *autentificados* para que se les *autorice* acceso de escritura (Andress, 2014: 40). La firma de llaves usualmente involucra la presentación en persona de una identificación emitida por un gobierno (un pasaporte, por ejemplo) y el chequeo de una llave criptográfica que puede ser utilizada para acceso futuro (Coleman, 2012: 143)[16]. Cambios tanto en los repositorios *Git* como

[15] Debian: The FTP Master Team. https://ftp-master.debian.org/

[16] Paso 5 de la firma de llaves: entregar la huella de tu llave. Debian Wiki: https://wiki.debian.org/Keysigning. Ver también los siguientes emails, públicamente archivados, y dirigidos a los a los nuevos mantenedores del listado de emails de Debian: (1) Steve Langasek – "Re: AM Report for Vagrant Cascadian vagrant@freegeek.org": https://lists.debian.org/debian-newmaint/2009/07/msg00043.html; (2) Joerg Jaspert – "Legal names (was: AM Report for Vagrant Cascadian vagrant@freegeek.org)": https://lists.debian.org/debian-newmaint/2009/07/msg00044.html. Asimismo, el paso dos del *Debian Developer's Corner: How You Can Join*: "se necesita verificar que la llave pertenece al Aplicante. Para lograr esto, la llave pública misma debe estar firmada por dos miembros de Debian. Por lo tanto, el Aplicante debe juntarse con este miembro de Debian en persona e identificarse (proveyendo un pasaporte, una licencia de conducir o algún otro tipo de ID)": https://www.debian.org/devel/join/nm-step2

en los paquetes *Debian* pueden ser – y, de acuerdo con las *best practices*, deben ser – firmados criptográficamente usando las llaves de desarrolladores particulares[17]. Si bien el sistema es notoriamente engorroso, en la práctica parece funcionar bien, al menos yo no he podido encontrar ninguna pista de intentos exitosos que comprometan estas llaves. Este sólido proceso contribuye al excelente historial de seguridad y reputación de *Debian*. Es por ello que las distribuciones basadas en *Debian* son probablemente las más populares en Linux, para los segmentos de mercado de escritorio, nubes o servidores (Vaughab-Nichols, 2018). Esto significa que está llegando a millones de usuarios.

La firma sirve como un precinto que enlaza un cambio particular en el software con una persona específica y que complementa el sello de tiempo. Los programas que manejan los repositorios para los usuarios finales deben – y regularmente lo hacen – comprobar dichas firmas para programáticamente establecer autenticidad. Los *Masters* son responsables de supervisar que todos los mantenedores y desarrolladores sigan los procedimientos acordados respecto a firmar cambios en *Git* y actualizaciones en paquetes *Debian*. La automatización ayuda mientras que todo esté llevándose a cabo de acuerdo a los planes, pero se quiebra inmediatamente en cuanto aparece la más mínima anomalía. Es sumamente difícil establecer la disciplina de quienes trabajan y encontrar las herramientas perfectas para un proyecto tecnológico de alta complejidad que depende fundamentalmente de trabajo voluntario distribuido. Nada expresa mejor el carácter moral que fundamenta dichas infraestructuras digitales que el comando de la *culpa* en los repositorios *Git*, el que permite, a quienes lo deseen, mirar quién hizo el último cambio en una línea particular del código fuente[18].

De manera similar a los mantenedores de *Debian*, quienes mantienen repositorios *Git* deben enfrentar el clásico desafío del archivo:

[17] Git Tools. Signing Your Work: https://git-scm.com/book/en/v2/Git-Tools-Signing-Your-Work
[18] Ver, documentación official de Git: https://git-scm.com/docs/git-blame

preservar y proveer la versión auténtica del software – pero en un panorama tecnológico rápidamente cambiante. Mientras que las versiones originales de los diarios de la escritora Sylvia Plath pueden decir la verdad acerca de su obra (Velody, 1998: 7-9), los usuarios del navegador Firefox no aceptarán la versión 1.0 como la versión auténtica, real o verdadera de éste por el simple hecho que el software está continuamente desarrollándose y adaptándose a su volátil contexto tecnológico. Tener siempre a mano una versión del software que los usuarios finales de sistemas puedan instalar, implica un trabajo de Sísifo de mantención y reparación vinculado al manejo del cambio sobre el tiempo; pese a ello, en esta tarea perdura el desafío de archivo que busca proveer acceso al objeto auténtico.

Conclusiones

Los grandes repositorios de software invierten y complican el concepto de autenticidad que es central a las prácticas de archivo. Mientras que la práctica clásica de archivo se organiza en torno a la preservación del documento original en cuanto versión auténtica, la mantención del repositorio considera la versión *reciente* del software auténtico. Téngase presente que "reciente" no necesariamente significa "último": en este sentido, el repositorio en cuanto archivo aún está interesado en el pasado, así como en la continua reparación y mantención de un contexto que lo vincula al presente en una manera específicamente significativa y potencialmente útil. La diferencia fundamental es que los archivos administran documentos estabilizados, mientras que los repositorios manejan *cambios* en el código.

Estas nociones tienen interesantes implicancias para el trabajo de archivo. Los repositorios no socavan sino destacan el interés clásico por los procedimientos institucionalizados, la confianza basada en la reputación y la contribución que hace el trabajo humano al archivo. Comparado con el

archivo, el repositorio es algo más útil y relevante que un contenedor pasivo de documentos auténticos importantes. Los mantenedores, desarrolladores y *FTP masters* de *Debian* trabajan día a día, desarrollando procedimientos estandarizados en contextos sociotécnicos automatizados, burocráticos y ritualizados que son impulsados por el intenso ethos ideológico-profesional de la meritocracia. La consistencia de sus esfuerzos por empacar software reciente es lo que establece la reputación del repositorio y su valor para los usuarios. La clave es la gestión de temporalidades auténticas compuestas por cambios en el software.

Doron Swade sostiene que en la era de los medios digitales, "en términos arqueológicos, no se puede asegurar la continuidad operacional de la cultura contemporánea" (Swade, en Ernst, 2013: 93). Por el contrario, a lo largo del presente texto he mostrado que quienes mantienen los repositorios enfrentan esta amenaza existencial a la modernidad como un desafío práctico más que como una condición ontológica. Esto se debe a que los repositorios requieren de trabajo continuo y creatividad diagramática para sostener su autenticidad. Temporalidades intrincadas son ensambladas a partir de cambios puntuales para incluir diagramáticamente y expresar de manera auténtica la peculiar resistencia (o liquidez) de las materialidades digitales. En este sentido, Chun llama a esta volatilidad del desarrollo del software lo "imperecedero efímero – una batalla de diligencia entre lo que pasa y lo repetitivo" (Chun, 2008: 167).

Ernst diagnostica cómo crecientemente los archivos analógicos se transforman en infraestructuras digitales definidas por protocolos diagramáticos más que por sus contenidos (Ernst, 2013: 84-85). El giro infraestructural cambia el énfasis desde "una enfática memoria cultural del largo plazo" hacia la "inmediata reproducción y reciclaje" (*ibíd*.: 95). Así, los repositorios de software son archivos que pierden su rol como una memoria cultural pues no están orientados hacia la eternidad (*ibíd*.: 100).

La investigación empírica sobre grandes repositorios de software y sus prácticas de mantención complica la difundida afirmación, repetida por

Ernst, que el medio microtemporal colapsa la continuidad del tiempo. Los repositorios están, en efecto, compuestos por cambios puntuales que pertenecen a momentos precisos indexados con un sello temporal. Pero el propósito mismo del trabajo humano que inserta software en repositorios orquestados es buscar que eventos mediáticos micro-temporales, tales como realizar cambios en el código del software, puedan ser ensamblados en formas de memoria cultural de largo plazo y en alta resolución.

Existen proyectos, como el de *Software Heritage*[19], que toman la aproximación clásica del archivista de "coleccionar, preservar y compartir" software y tienen los problemas de la disponibilidad, trazabilidad y uniformidad como problemas fundamentales. Incidentalmente, unas de sus principales fuentes de colección son los repositorios *Debian* y *Git*[20]. Mientras que los proyectos de archivo explícitamente construyen infraestructuras digitales orientadas hacia la "memoria cultural enfática" que Ernst tanto añora en el paisaje mediático, yo me he focalizado en los repositorios en tanto archivos. Los repositorios desarraigan el pesimismo tecnológico-determinista de Ernst de maneras más interesantes. En tanto son utilizados para confrontar la temporalidad volátil del software, los repositorios emplean un abordaje de archivo para mantener las infraestructuras digitales de las cuales usuarios y desarrolladores dependen en su vida cotidiana.

En conclusión, los repositorios son archivos de cambios. Archivar cambios antes que documentos requiere de un repensar las prácticas de archivo junto con sus líneas infraestructurales. Aun así, las preocupaciones nucleares del archivo, relativas a la autenticidad y la preservación, son hoy más, antes que menos, centrales para la estructuración de las relaciones sociales en la era de la reproducción mecánica.

[19] Para la misión y aproximación del Software Heritage, ver: https://www.softwareheritage.org/mission/ y https://www.softwareheritage.org/mission/approach/. Para proyectos similares enfocados solo en *Debian*, ver: http://snapshot.debian.org/ y http://archive.debian.org/README

[20] Ver, respectivamente, el *Debian package loader* y el *Git loader*: https://forge.softwareheritage.org/source/swhloader-debian/ y https://forge.softwareheritage.org/source/swh-loader-git/

Nota

Todas las figuras ocupadas en este artículo poseen licencia *creative commons*. La figura 1 posee licencia 'Creative Commons Attribution 2.5 Australia'; la figura 2, 'Creative Commons Attribution–NonCommercial-ShareAlike 3.0 unported'.

Agradecimientos

Me gustaría agradecer a la extensa comunidad de software libre, en particular, a los desarrolladores *Debian*, por sus trabajos voluntarios de mantención y desarrollo, así como por su apoyo a los usuarios. *anarcat*, *dkg*, *efkin* y *pabs* son algunos de los hackers que me han entregado útiles observaciones sobre los aspectos técnicos de este artículo. Agradezco a los editores por la invitación a participar de este número especial y a Francisco Salinas por su generosidad al traducir este texto desde el inglés. Finalmente, el artículo de David Murphy "Hacking Public Memory" (2013) y su bibliografía encendió mi interés en los repositorios como archivos.

Sobre el autor

Maxigas es profesor en práctica crítica de medios digitales en el Departamento de Sociología de la Universidad de Lancaster, dónde también es miembro del *Centre for Science Studies*. Sus intereses investigativos son el rol de la cibernética clásica en la formación de imaginarios tecno-políticos y la imaginación teórico-social; la historia y geografía de las prácticas de *hacking*, con un foco particular en las trayectorias europeas; cómo los innovadores usan tecnologías antiguas ("the Luddite aspects of hackerdom"), incluyendo la historia social y el uso contemporáneo del Internet Relay Chat. Ha publicado extensamente en talleres compartidos [*Shared Machine Shops*] y es parte del comité editorial del *Journal of Peer Production*.

Referencias

Akrich, Madeleine (1992). "The de-Scription of Technical Objects", en Wiebe Eco Bijker y John Law (Eds.) *ShapingTechnology / Building Society: Studies in Sociotechnical Change*. Cambridge, MIT Press: 205–224.

Andress, Jason (2014). *The Basics of Information Security: Understanding the Fundamentals of Infosec in Theory and Practice*. Segunda edición. Waltharn.

Beck, Ulrich, Wolfgang Bons y Christoph Lau (2003). "The Theory of Reflexive Modernization", en *Theory, Society and Culture* 20 (2): 1–33. doi:10.1177/0263276403020002001.

Chun, Wendy Hui Kyong (2016). *Updating to Remain the Same: Habitual New Media*. First edition. Cambridge, MA, MIT Press.

Chun, Wendy Hui Kyong (2011). *Programmed Visions: Software and Memory*. Computer software Studies. Cambridge, MA, MIT Press.

Chun, Wendy Hui Kyong (2008). "The Enduring Ephemeral, or the Future Is a Memory", en *Critical Inquiry* 35 (1): 148–171.

Coleman, Gabriella (2012). *Coding Freedom: The Ethics and Aesthetics of Hacking*. Princeton, Princeton University Press.

Dafermos, George (2012). *Governance Structures of Free/Open Source Software Development: Examining the Role of Modular Product Design as a Governance Mechanism in the FreeBSD Project*. NGInfra Phd Thesis Series on Infrastructures 51. Delft, Next Generation Infrastructures Foundation.

Deleuze, Gilles y Félix Guattari (1987). *A Thousand Plateaus*. Minnesota, MN, University of Minnesota Press.

Ernst, Wolfgang. 2016. *Chronopoetics: The Temporal Being and Operativity of Technological Media*. Media Philosophy. Londres, Rowman & Littlefield International.

Ernst, Wolfgang (2013). *Digital Memory and the Archive*. Electronic Mediations 39. Minneapolis, University of Minnesota Press.

Ernst, Wolfgang (2005). "Art of the Archive", en Helen Adkins (ed.) *Künstler.Archiv: Neue Werke Zu Historichen Beständen*. Köln, Walter König.

Featherstone, Mike (2006). "Archive", en *Theory, Culture, Society* 23 (2-3): 591–596.

Fuller, Matthew (2003). *Behind the Blip. Essays on the Culture of Software*. New York, Autonomedia. Disponible en: http://libgen.io/book/index.php?md5=53B9E3428B4E41ECEB3A758F87B8A035.

Kamini, Vellodi (2012). *Tintoretto's Difference: Deleuze, Diagrammatics,*

and the Problem of Art History. Tesis de Doctorado. Hendon, Middlesex University. Disponible en: http://www.kaminivellodi.com/wp-content/uploads/2015/11/Tintorettos-Time.pdf

Maxigas (2012). *This Is R0ket Science!: Modernity, Capitalism, Liberalism and Hacker Culture*. Tesis de Magister en Sociología y Antropología Social. Budapest, Central European University. Disponible en: http://www.etd.ceu.hu/2012/dunajcsik_peter.pdf.

Maxigas (2017). "Hackers Against Technology: Critique and Recuperation in Technological Cycles", en *Social Studies of Science* 30: 841-860. doi:10.1177/0306312717736387.

Murphy, David (2013). "Hacking Public Memory: Understanding the Multiple Arcade Machine Emulator", en *Games and Culture* 8 (43): 43–54.

Steinlin, Guadenz (2009). "Cooperation and Social Status in Free and Open Source Projects: Results from an Empirical Study of the Debian Project." Presentación en *DebConf9*. Disponible en: http://penta.debconf.org/dc9_schedule/events/456.en.html

Vaughan-Nichols, Steven J. (2018). "What's the Most Popular Linux of Them All?: The Data's Messy for Picking Out the Most Popular Linux Distributions, but There Are Some Things Know We Know." *ZDnet article*. Disponible en: https://www.zdnet.com/article/whats-the-most-popular-linux-of-them-all/

Velody, Irving (1998). "The Archive and the Human Sciences: Notes Towards a Theory of the Archive", en *History of the Human Sciences* 11 (4): 1–16.

Virilio, Paul (2006). *Speed and Politics*. Semiotext(e) Foreign Agents Series. Cambridge, MA, MIT Press.

Traducido por Francisco Salinas