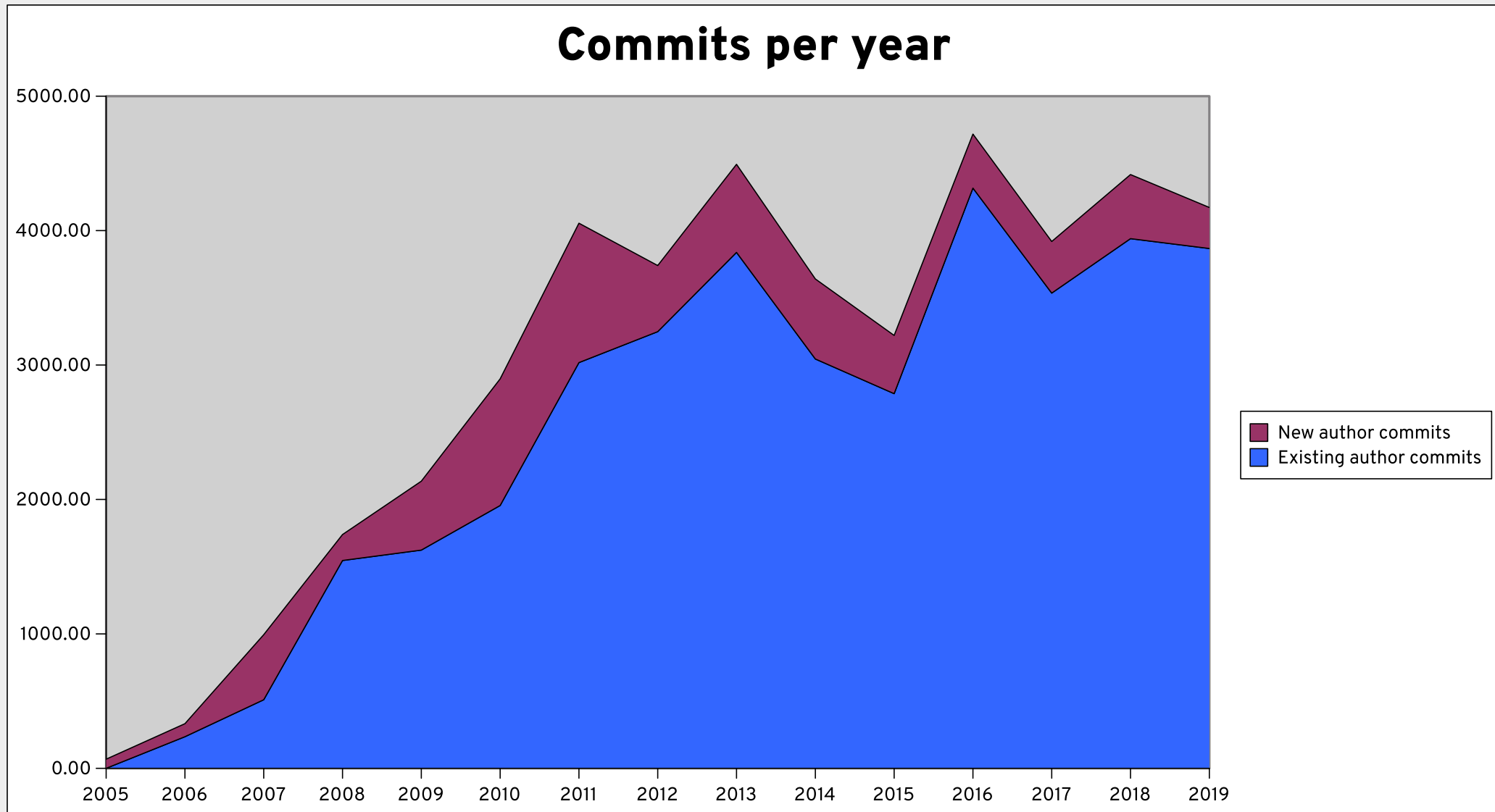




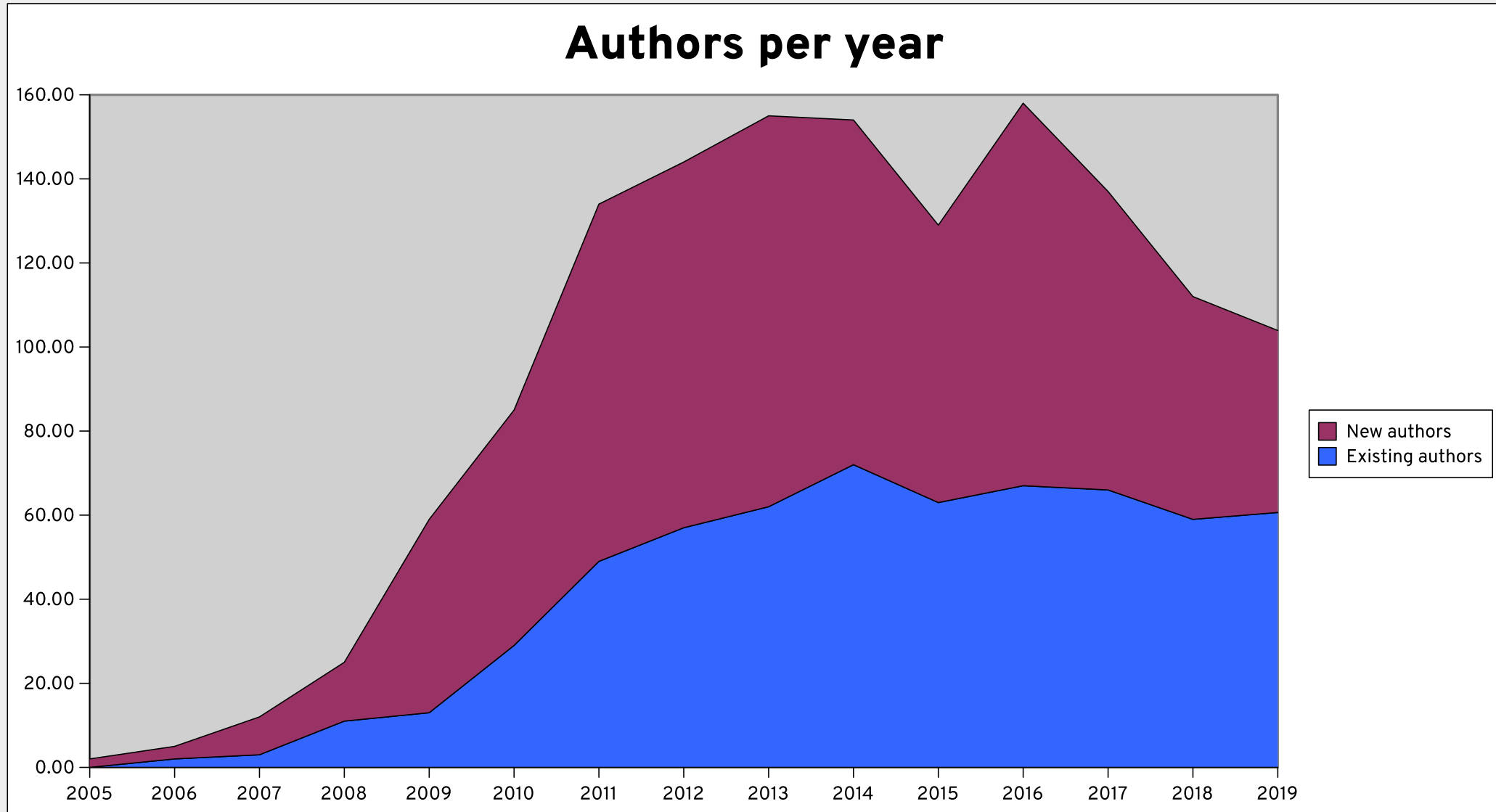
Never too late to learn new tricks

Daniel P. Berrangé
@
KVM Forum 2019, Lyon

Then: The last 14 years



Then: The last 14 years



Now: Virt usage is changing

- **Libvirt architecture stable for 12+ years**
 - libvirtd, C language, handle OOM, GNULIB
 - Refactoring is evolutionary not revolutionary
- **Libvirt historically used with**
 - Data center virt (oVirt)
 - Public/private cloud (OpenStack)
 - Desktop virt (Boxes, virt-manager)
- **New ideas/architecture concepts**
 - KVM inside containers (Kubernetes / KubeVirt)
 - KVM outside containers (Kata Containers)
 - MicroVMs for functions (Firecracker)



Now: libvirt adaptation

- **libvirt must be sustainable long term**
 - Attractive work/challenges for new contributors
 - Attractive features for application developers
- **Contributors must work efficiently**
 - Less time on grunt work / recreating wheels
 - More time on **features that matter** to apps



Now: Modular daemons

- **libvirtd, all drivers in one process**
 - One driver can break all, no security isolation
 - Also provides remote IP access
- **vir\${DRV}d, one driver per process**
 - virtqemud, virtxend, virtnetworkd, virtstoraged, virt...d
 - virtproxyd, remote IP access
- **Libvirtd still default, switching in 2020**
 - App APIs stable, deployment tools impacted



Just now: ENOMEM handling

- **Linux malloc() (mostly) doesn't fail**
- **Complexity from goto cleanup jumps**
 - Difficult to test thoroughly
 - App code doesn't handle ENOMEM
- **Switch to abort() on ENOMEM**
- **Reduces burden for libvirt maintainers**



Just now: Automatic cleanup

- **Libvirt already mandates GCC/Clang**
 - Can leverage C extensions from
- **`__attribute__((cleanup(func))) type var;`**
 - Run 'func' when 'var' goes out of scope
 - Eliminates majority of explicit free() calls
 - Reduces memory leaks & code complexity
 - Also close fd, unref object, unlock mutex
- **Reduces burden for libvirt maintainers**



Now: GLib, C's std library

- **Libvirt written to POSIX API “standard”**
 - Poor OS compliance, much optional
 - Very low level, painful for direct use
- **GNULIB papers over many differences**
 - Tied to autotools with complex bootstrap
- **Libvirt adds many higher level APIs**
 - Re-invents the wheel vs many other C apps/libs



Now: GLib, C's std library

- **GLib is a high level C “std lib”**
 - Many data structures
 - Event loop impl
 - Objects for sockets, I/O, DBus services/clients
 - Introspection for language bindings
- **Previously avoided due to ENOMEM abort()**
- **Reduces burden for libvirt maintainers**
 - Less worrying about portability / no more GNULIB
 - More time on interesting virt features



Now: Language consolidation

- **Libvirt is written in C**
 - Hey, what's all this python, perl, shell, xsl, html, markdown, m4, make, automake, sed, awk...
- **Reduce knowledge burden on contributors**
 - One language for each job
- **shell, sed, awk, perl → python**
 - Broader developer talent pool
- **shell, sed, awk, make, m4, automake → meson**
 - Attractive/simpler DSL for build tools
- **XSL, HTML, Markdown → RST (w/ Pelican/Sphinx ?)**
 - Simpler markup language & templating system



Next: Embedded QEMU driver

- **Libvirt design suits traditional virt usage**
- **VMs can be used as an service technology**
 - eg libguestfs spawns a QEMU appliance
- **Use cases shouldn't interfere**
- **New driver mode “qemu:///embed”**
 - No libvirtd involved
 - Runs inside app process
 - Invisible to other libvirt clients
 - Isolated to private directory subtree



Then: Memory unsafe languages

<https://msrc-blog.microsoft.com/2019/07/16/a-proactive-approach-to-more-secure-code/>

- **~70% of all reported CVEs are memory safety bugs in C/C++ code**
 - Use after free, stack smashing, heap corruption
 - Despite many analysis tools & better compilers, CVE rate has not improved in 12+ years
 - Memory safety CVEs are preferentially exploited
- **Writing C/C++ without memory mgmt errors is not possible**



Next: Memory safe languages

- **Common languages had too many caveats for systems programming**
 - Java – JVM memory footprint
 - Python – limited performance / thread scaling
- **Rust & Golang change the situation**
 - Perf/footprint close to/matching/exceeding C
 - Golang has simplicity of a language like Python
 - Young but rapidly growing / maturing ecosystem
- **C is no longer the sensible choice**



Next: Memory safe languages

- **libvirt to integrate Rust, Golang**
 - Still TBD which (one, other, both)
- **Targeted adoption in existing code**
 - Rewriting has costs (stability, reviewer time)
 - Benefits must outweigh cost
- **Long term effort**
 - Conversion work will last 5, 10, 1729,... years
 - Still need to deliver user features effectively



Then: Autotools build system

- **Many languages needed**
 - shell, sed, awk, m4, make, autoconf, automake, python, perl, and more
 - Large burden for new contributors to learn
 - Many poorly understood even by regular contributors
- **‘configure’ a 1.8 MB, very slow shell script**
 - Increasingly dominates build time, can’t be parallelized



Next: Meson build system

- **A self contained DSL for build rules**
 - Well documented, simple to understand syntax
 - Call out to python if needed
 - Active & responsive upstream for RFEs
 - Sensible defaults (parallel build, compiler flags on error)
- **Older distros may have to bundle meson**
 - Better than bundling 1.8 MB autoconf shell script



Now: mailing list code review

- **New contributor patch submit pitfalls**
 - HTML mail instead of plain text
 - Mangled patches from mail client
 - Incorrectly threaded series
 - Not labelling series with version numbers
 - Sending plain 'diff' output instead of a git patch
 - Subscribe to yet another service
 - Not basing on git master
 - Corporate legal privacy / copyright signatures
 - Unintelligible mail quoting in replies (Outlook)
 - DMARC/DKIM problems withg mailing lists
- **First impressions matter for new contributors**



Next: Web based code review

- **New contributor familiarity**
 - More widely used than mailing lists
 - Fewer ways to mess up patch submission
 - More easily see outstanding submissions
- **Remote API service**
 - Rich metadata for analysis / reporting
 - Build custom tools to ease development
- **Ties into bug tracking & CI services**
- **Plan is still TBD...watch this space**





FIN