White Paper: Red Hat Crash Utility

by David Anderson
<anderson@redhat.com>

Content

Abstract

The Red Hat `crash` analysis utility is loosely based on the SVR4 UNIX `crash` command, but has been significantly enhanced by completely merging it with the GNU `gdb` debugger. The marriage of the two effectively combines the kernel-specific nature of the traditional UNIX `crash` utility with the source code level debugging capabilities of `gdb`. The utility can be used to investigate:

- Live Linux systems
- Linux kernel core dumps created by the Kdump facility
- Compressed Linux kernel core dumps created by the makedumpfile command (from kdump dumpfiles)
- Linux kernel core dumps created from by the Red Hat Netdump facility
- Linux kernel core dumps created from by the Red Hat Diskdump facility
- Compressed Linux kernel core dumps created by the Red Hat Diskdump facility
- Xen host Linux kernel core dumps created by the Kdump facility
- Xen guest Linux kernel core dumps created by the original xendump facility
- Xen guest Linux kernel core dumps created by the ELF-format xendump facility
- Xen hypervisor core dumps created by the Kdump facility

- KVM guest Linux kernel core dumps created by the `virsh dump` facility
- s390 Linux kernel core dumps created by the IBM standalone core dump facility.
- s390x Linux kernel core dumps created by the IBM standalone core dump facility.
- Linux kernel core dumps created by the LKCD (Linux Kernel Crash Dumps) Sourceforge project
- Linux kernel core dumps created by the Mcore patch offered by Mission Critical Linux

The current set of commands consist of common kernel core analysis tools such as kernel stack back traces of all processes, source code disassembly, formatted kernel structure and variable displays, virtual memory data, dumps of linked-lists, etc., along with several commands that delve deeper into specific kernel subsystems. Relevant `gdb` commands may also be entered, which in turn are passed on to the `gdb` module for execution.

The `crash` utility is designed to be independent of Linux version dependencies. When new kernel source code impacts the correct functionality of `crash` and its command set, the utility will be updated to recognize new kernel code changes while maintaining backwards compatibility with earlier releases. The most current version of the `crash` utility may be found here: http://people.redhat.com/anderson

---

Why Crash?

The Linux operating system originally lacked a built-in, traditional UNIX-like kernel crash dump facility. This was initially addressed by the Mission Critical Linux Mcore kernel patch and the LKCD (Linux Kernel Crash Dump) kernel patch from SGI in 1999, and later by the Red Hat Netdump facility in 2002, and the Red hat Diskdump facility in 2004. The upstream Linux community finally settled upon the adoption of the Kdump crash dump facility in 2006.

However, the creation of a kernel crash dump file is only half of the picture; a utility is required to be able to recognize the dumpfile format in order to read it, and to offer a useful set of commands to make sense of it.

Furthermore, to examine the contents of a live system's kernel internals from user space, the only readily available option has been to use `gdb` on `/proc/kcore`. While `gdb` is an incredibly powerful tool, it is designed to debug user programs, and is not at all "kernel-aware". Consequently, using `gdb` alone has limited usefulness when looking at kernel memory, essentially constrained to the printing of kernel data structures ***if*** the `vmlinux` file was built with the `-g` C flag, the disassembly of kernel text, and raw data dumps. Furthermore, distributions such as Red Hat Enterprise Linux have limited the access to `/proc/kcore`, making it unusable as a kernel memory source.

As far as kernel crash dump files are concerned, the Red Hat Netdump and uncompressed Diskdump facilities, and the Kdump facility create dump files that ***are*** readable by `gdb`, but aside from giving it the capability of displaying the panicking task's stack trace, it has the same constraints as when reading `/proc/kcore`. However, `gdb` cannot read LKCD, Mcore, Xen or s390/s390x dump files.

That being the state of things, the `crash` utility was developed as a convenient means to cover all bases, i.e., all listed dumpfile formats as well as live systems. Moreover, it is also designed to be easily enhanced to suit the specific needs of the kernel developers or analysts using it; the builtin command set can easily be extended or enhanced, and

external command modules may be written and dynamically attached.

Prerequisites

The `crash` utility has the following prerequisites:

kernel object file:

A `vmlinux` kernel object file, often referred to as the *namelist* in this document, which ***must*** have been built with the `-g` C flag so that it will contain the debug data required for symbolic debugging.

In RHEL3 installations, the `vmlinux` file associated with the running kernel is split into two files, a stripped version found in the `/boot` directory; which has have the operating system release string appended to it, for example, `vmlinux-2.4.21-4.ELsmp`. The stripped file in `/boot` contains a link to its associated debuginfo file, which is located in the `/usr/lib/debug/boot` directory.

In RHEL4, RHEL5 and RHEL6 installations, the `vmlinux` file is part of the kernel debuginfo package, and is found in the relevant `/usr/lib/debug/lib/modules/<release>` directory.

Ideally the kernel object file is the same kernel object file that is associated with the memory image. However, in circumstances where the `vmlinux` file associated with the crash dump or live system was ***not*** built with the `-g` flag, there are work-arounds discussed later in the Invocation section.

memory image:

This may consist of a kernel crash dump file generated from any of the supported dump facilties, or live system memory accessed via `/dev/mem` or its replacement in RHEL4/RHEL5/RHEL6, the `/dev/crash` driver. If no dump file argument is issued on the `crash` command line, live system memory will be used by default. When examining a live system, root privileges are required.

platform processor types:

The `crash` utility is actively developed and tested on the x86, x86_64, ia64, ppc64, arm, s390 and s390x processors. Legacy support for the Alpha and 32-bit PowerPC platforms exists, but no longer actively maintained.

Linux kernel versions:

The `crash` utility is backwards-compatible to at least Red Hat 6.0 (Linux version 2.2.5-15), up to Red Hat Enterprise Linux 5 (Linux version 2.6.18+). Due to the constantly shifting sands of the upstream kernel internals, immediate support for the latest kernel versions cannot be guaranteed.

However, modifications are constantly being implemented to support changes in upstream kernel versions. The intent has always been to make the utility independent of Linux version dependencies, building in recognition of major kernel code changes so as to adapt to new kernel versions, while maintaining backwards compatibility.

Build Procedure

Starting with the RHEL3 release, the `crash` utility is automatically installed during system installation if the **Development Tools** package set is selected. However, for all other kernel versions, or if it was not selected during system installation, the binary RPM can be installed, or if desired, the sources re-built and installed.

If the `crash` utility is not pre-installed, and if all dependencies are met on the target system, install the binary RPM like so:

```
# rpm -ivh crash-4.0-8.11.i386.rpm
Preparing...                ########################################### [100%]
   1:crash                  ########################################### [100%]
#
```

The `crash` executable will be installed in the `/usr/bin` directory.

Alternatively, the `crash` source code can be rebuilt. The `crash` utility's source files come packaged in two typical formats, a compressed tar image. or a source RPM file, So, for example, `crash` version 4.0-8.11 can be built from either `crash-4.0-8.11.tar.gz` or `crash-4.0-8.11.src.rpm`.

The latest "upstream" version of the `crash` utility, available in both file formats, can be found here: http://people.redhat.com/anderson

In either case, the source file layout consists of a top-level directory containing a set of `crash`-specific files, a compressed tar image containing the full, unmodified, `gdb` source tree, and a small number of modified `gdb` files required to merge the two entities. The build procedure does the following:

1. the unmodified `gdb` sources are extracted into a subdirectory of the top-level `crash` source directory, and overlayed by the small set of modified `gdb` files.
2. the files in the `gdb` source tree are built first, creating the `libbfd.a`, `libreadline.a`, `libopcodes.a`, `libiberty.a` and `libgdb.a` libraries.
3. the `crash` sources files in the top-level directory are then compiled into a `crashlib.a` library.
4. the objects are all linked into the `crash` executable, located in the top-level directory.

Depending upon the speed of the host system, the complete build may take several minutes, primarily due to the time consumed by the build of the `gdb` portion.

Building from the tar image

To build from the compressed tar image, simply uncompress/extract the source files, `cd` into the resultant source directory, and enter `make`:

```
# tar xvzmf crash-4.0-8.11.tar.gz
crash-4.0-8.11/
crash-4.0-8.11/main.c
crash-4.0-8.11/tools.c
crash-4.0-8.11/global_data.c
crash-4.0-8.11/memory.c
crash-4.0-8.11/filesys.c
crash-4.0-8.11/help.c
crash-4.0-8.11/task.c
crash-4.0-8.11/kernel.c
crash-4.0-8.11/test.c
crash-4.0-8.11/gdb_interface.c
crash-4.0-8.11/configure.c
crash-4.0-8.11/net.c
crash-4.0-8.11/dev.c
crash-4.0-8.11/alpha.c
crash-4.0-8.11/x86.c
crash-4.0-8.11/ppc.c
crash-4.0-8.11/ia64.c
crash-4.0-8.11/s390.c
crash-4.0-8.11/s390x.c
crash-4.0-8.11/s390dbf.c
crash-4.0-8.11/ppc64.c
crash-4.0-8.11/x86_64.c
crash-4.0-8.11/extensions.c
crash-4.0-8.11/remote.c
crash-4.0-8.11/va_server.c
crash-4.0-8.11/va_server_v1.c
crash-4.0-8.11/symbols.c
crash-4.0-8.11/cmdline.c
crash-4.0-8.11/lkcd_common.c
crash-4.0-8.11/lkcd_v1.c
crash-4.0-8.11/lkcd_v2_v3.c
crash-4.0-8.11/lkcd_v5.c
crash-4.0-8.11/lkcd_v7.c
crash-4.0-8.11/lkcd_v8.c
crash-4.0-8.11/lkcd_fix_mem.c
crash-4.0-8.11/s390_dump.c
crash-4.0-8.11/lkcd_x86_trace.c
crash-4.0-8.11/netdump.c
crash-4.0-8.11/diskdump.c
crash-4.0-8.11/xendump.c
crash-4.0-8.11/unwind.c
crash-4.0-8.11/unwind_decoder.c
crash-4.0-8.11/unwind_x86_32_64.c
crash-4.0-8.11/xen_hyper.c
crash-4.0-8.11/xen_hyper_command.c
crash-4.0-8.11/xen_hyper_global_data.c
crash-4.0-8.11/xen_hyper_dump_tables.c
crash-4.0-8.11/defs.h
crash-4.0-8.11/xen_hyper_defs.h
crash-4.0-8.11/va_server.h
crash-4.0-8.11/vas_crash.h
crash-4.0-8.11/netdump.h
crash-4.0-8.11/diskdump.h
crash-4.0-8.11/xendump.h
crash-4.0-8.11/unwind.h
crash-4.0-8.11/unwind_i.h
crash-4.0-8.11/rse.h
crash-4.0-8.11/unwind_x86.h
crash-4.0-8.11/unwind_x86_64.h
crash-4.0-8.11/lkcd_vmdump_v1.h
crash-4.0-8.11/lkcd_vmdump_v2_v3.h
crash-4.0-8.11/lkcd_dump_v5.h
crash-4.0-8.11/lkcd_dump_v7.h
crash-4.0-8.11/lkcd_dump_v8.h
crash-4.0-8.11/lkcd_x86_trace.h
crash-4.0-8.11/lkcd_fix_mem.h
crash-4.0-8.11/ibm_common.h
crash-4.0-8.11/Makefile
crash-4.0-8.11/gdb-6.1/
crash-4.0-8.11/gdb-6.1/gdb/
crash-4.0-8.11/gdb-6.1/gdb/Makefile.in
crash-4.0-8.11/gdb-6.1/gdb/main.c
crash-4.0-8.11/gdb-6.1/gdb/symtab.c
```

```
crash-4.0-8.11/gdb-6.1/gdb/target.c
crash-4.0-8.11/gdb-6.1/gdb/symfile.c
crash-4.0-8.11/gdb-6.1/gdb/elfread.c
crash-4.0-8.11/gdb-6.1/gdb/ui-file.c
crash-4.0-8.11/gdb-6.1/gdb/utils.c
crash-4.0-8.11/gdb-6.1/gdb/dwarf2read.c
crash-4.0-8.11/gdb-6.1/gdb/ppc-linux-tdep.c
crash-4.0-8.11/gdb-6.1/Makefile.in
crash-4.0-8.11/gdb-6.1/include/
crash-4.0-8.11/gdb-6.1/include/obstack.h
crash-4.0-8.11/gdb-6.1.patch
crash-4.0-8.11/COPYING
crash-4.0-8.11/.rh_rpm_package
crash-4.0-8.11/crash.8
crash-4.0-8.11/extensions/
crash-4.0-8.11/extensions/Makefile
crash-4.0-8.11/extensions/echo.c
crash-4.0-8.11/extensions/dminfo.c
crash-4.0-8.11/extensions/libsial/
crash-4.0-8.11/extensions/libsial/Makefile
crash-4.0-8.11/extensions/libsial/mkbaseop.c
crash-4.0-8.11/extensions/libsial/README
crash-4.0-8.11/extensions/libsial/README.sial
crash-4.0-8.11/extensions/libsial/sial_alloc.c
crash-4.0-8.11/extensions/libsial/sial_api.c
crash-4.0-8.11/extensions/libsial/sial_api.h
crash-4.0-8.11/extensions/libsial/sial_builtin.c
crash-4.0-8.11/extensions/libsial/sial_case.c
crash-4.0-8.11/extensions/libsial/sial_define.c
crash-4.0-8.11/extensions/libsial/sial_func.c
crash-4.0-8.11/extensions/libsial/sial.h
crash-4.0-8.11/extensions/libsial/sial_input.c
crash-4.0-8.11/extensions/libsial/sial.l
crash-4.0-8.11/extensions/libsial/sial-lsed
crash-4.0-8.11/extensions/libsial/sial_member.c
crash-4.0-8.11/extensions/libsial/sial_node.c
crash-4.0-8.11/extensions/libsial/sial_num.c
crash-4.0-8.11/extensions/libsial/sial_op.c
crash-4.0-8.11/extensions/libsial/sialpp.l
crash-4.0-8.11/extensions/libsial/sialpp-lsed
crash-4.0-8.11/extensions/libsial/sialpp.y
crash-4.0-8.11/extensions/libsial/sial_print.c
crash-4.0-8.11/extensions/libsial/sial_stat.c
crash-4.0-8.11/extensions/libsial/sial_str.c
crash-4.0-8.11/extensions/libsial/sial_type.c
crash-4.0-8.11/extensions/libsial/sial_util.c
crash-4.0-8.11/extensions/libsial/sial_var.c
crash-4.0-8.11/extensions/libsial/sial.y
crash-4.0-8.11/extensions/sial.c
crash-4.0-8.11/extensions/sial.mk
crash-4.0-8.11/gdb-6.1.tar.gz
crash-4.0-8.11/README
# cd crash-4.0-8.11
# make
TARGET: X86
 CRASH: 4.0-8.11
   GDB: 6.1
gdb-6.1/gdb/CONTRIBUTE
gdb-6.1/gdb/COPYING
gdb-6.1/gdb/ChangeLog
gdb-6.1/gdb/ChangeLog-1990
gdb-6.1/gdb/ChangeLog-1991
gdb-6.1/gdb/ChangeLog-1992
gdb-6.1/gdb/ChangeLog-1993
gdb-6.1/gdb/ChangeLog-1994
gdb-6.1/gdb/ChangeLog-1995
gdb-6.1/gdb/ChangeLog-1996
gdb-6.1/gdb/ChangeLog-1997
gdb-6.1/gdb/ChangeLog-1998
gdb-6.1/gdb/ChangeLog-1999
gdb-6.1/gdb/ChangeLog-2000
gdb-6.1/gdb/ChangeLog-2001
gdb-6.1/gdb/ChangeLog-2002
gdb-6.1/gdb/ChangeLog-2003
gdb-6.1/gdb/ChangeLog-3.x
```

```
gdb-6.1/gdb/MAINTAINERS
gdb-6.1/gdb/NEWS
gdb-6.1/gdb/PROBLEMS
gdb-6.1/gdb/README
gdb-6.1/gdb/TODO
gdb-6.1/gdb/abug-rom.c
gdb-6.1/gdb/acconfig.h
gdb-6.1/gdb/acinclude.m4
gdb-6.1/gdb/aclocal.m4
gdb-6.1/gdb/ada-exp.y
gdb-6.1/gdb/ada-lang.c
gdb-6.1/gdb/ada-lang.h

  (complete output not shown)

ar -rs crashlib.a main.o tools.o global_data.o memory.o filesys.o help.o task.o
build_data.o kernel.o test.o gdb_interface.o net.o dev.o alpha.o x86.o ppc.o ia6
4.o s390.o s390x.o s390dbf.o ppc64.o x86_64.o extensions.o remote.o va_server.o
va_server_v1.o symbols.o cmdline.o lkcd_common.o lkcd_v1.o lkcd_v2_v3.o lkcd_v5.
o lkcd_v7.o lkcd_v8.o lkcd_fix_mem.o s390_dump.o netdump.o diskdump.o xendump.o
lkcd_x86_trace.o unwind_v1.o unwind_v2.o unwind_v3.o unwind_x86_32_64.o xen_hype
r.o xen_hyper_command.o xen_hyper_global_data.o xen_hyper_dump_tables.o
ar: creating crashlib.a
gcc -g -O2       \
               -o `cat mergeobj` libgdb.a \
                  ../bfd/libbfd.a ../readline/libreadline.a ../opcodes/libopcod
es.a  ../libiberty/libiberty.a     -lm -lncurses  ../libiberty/libiberty.a  -ldl
 -rdynamic `cat mergelibs`
#
```

The resultant `crash` executable will be located in the top-level source directory. Install it in `/usr/bin` by entering:

```
# make install
/usr/bin/install crash /usr/bin
#
```

Building from the source RPM

To build from the source RPM, install the `crash-4.0-8.11.src.rpm`, `cd` to the appropriate `SPECS` directory, and build the package:

```
# rpm -Uvh crash-4.0-8.11.src.rpm
   1:crash              ########################################### [100%]
# cd /usr/src/redhat/SPECS
# rpmbuild -ba crash.spec
Executing(%prep): /bin/sh -e /var/tmp/rpm-tmp.19005
+ umask 022
+ cd /usr/src/redhat/BUILD
+ LANG=C
+ export LANG
+ unset DISPLAY
+ cd /usr/src/redhat/BUILD
+ rm -rf crash-4.0-8.11
+ /bin/gzip -dc /usr/src/redhat/SOURCES/crash-4.0-8.11.tar.gz
+ tar -xvvf -
drwxr-xr-x root/root         0 2002-09-12 16:51:40 crash-4.0-8.11/
-rwxrwxr-x root/root     31916 2002-09-12 16:51:40 crash-4.0-8.11/main.c
-rwxrwxr-x root/root    103454 2002-09-12 16:51:40 crash-4.0-8.11/tools.c
-rwxrwxr-x root/root      5802 2002-09-12 16:51:40 crash-4.0-8.11/global_data.c
-rwxrwxr-x root/root    225343 2002-09-12 16:51:40 crash-4.0-8.11/memory.c
-rwxrwxr-x root/root     75492 2002-09-12 16:51:40 crash-4.0-8.11/filesys.c
-rwxrwxr-x root/root    211519 2002-09-12 16:51:40 crash-4.0-8.11/help.c
-rwxrwxr-x root/root    110604 2002-09-12 16:51:40 crash-4.0-8.11/task.c
-rwxrwxr-x root/root    101805 2002-09-12 16:51:40 crash-4.0-8.11/kernel.c
-rwxrwxr-x root/root      2198 2002-09-12 16:51:40 crash-4.0-8.11/test.c
-rwxrwxr-x root/root     18949 2002-09-12 16:51:40 crash-4.0-8.11/gdb_interface.c
-rwxrwxr-x root/root     20239 2002-09-12 16:51:40 crash-4.0-8.11/configure.c
```

```
-rwxrwxr-x root/root     29931 2002-09-12 16:51:40 crash-4.0-8.11/net.c
-rwxrwxr-x root/root     99654 2002-09-12 16:51:40 crash-4.0-8.11/dev.c
-rwxrwxr-x root/root     76146 2002-09-12 16:51:40 crash-4.0-8.11/alpha.c
-rwxrwxr-x root/root     74638 2002-09-12 16:51:40 crash-4.0-8.11/x86.c
-rwxrwxr-x root/root     42109 2002-09-12 16:51:40 crash-4.0-8.11/ppc.c
-rwxrwxr-x root/root     76357 2002-09-12 16:51:40 crash-4.0-8.11/ia64.c

  (complete output not shown)

Requires: libc.so.6 libc.so.6(GLIBC_2.0) libc.so.6(GLIBC_2.1) libc.so.6(GLIBC_2.
2) libc.so.6(GLIBC_2.2.3) libc.so.6(GLIBC_2.3) libdl.so.2 libdl.so.2(GLIBC_2.0)
libdl.so.2(GLIBC_2.1) libm.so.6 libm.so.6(GLIBC_2.0) libncurses.so.5 libz.so.1
Processing files: crash-debuginfo-4.0-8.11
Requires(rpmlib): rpmlib(CompressedFileNames) <= 3.0.4-1 rpmlib(PayloadFilesHave
Prefix) <= 4.0-1
Checking for unpackaged file(s): /usr/lib/rpm/check-files /var/tmp/crash-root
Wrote: /usr/src/redhat/SRPMS/crash-4.0-8.11.src.rpm
Wrote: /usr/src/redhat/RPMS/i386/crash-4.0-8.11.i386.rpm
Wrote: /usr/src/redhat/RPMS/i386/crash-debuginfo-4.0-8.11.i386.rpm
#
```

Lastly, install the binary RPM, which copies the `crash` executable to the `/usr/bin` directory:

```
# rpm -ivh /usr/src/redhat/RPMS/i386/crash-4.0-8.11.i386.rpm
Preparing...                ########################################### [100%]
   1:crash
#
```

Invocation

When `crash` is run on a dumpfile, at least two arguments are always required:

1. The kernel object filename, often referred to as the kernel ***namelist***. When initially built from the kernel sources, its name is `vmlinux`. In RHEL3 installations, it is copied to the `/boot` directory, where the operating system release number is appended, as in `vmlinux-2.4.21-4.ELsmp`. In RHEL4, RHEL5 and RHEL6 installations, the `vmlinux` file is part of the kernel debuginfo package, and is located in the relevant `/usr/lib/debug/lib/modules/<release>` directory.
2. The dumpfile name, typically named `vmcore`.

For example, if both files are located in the current directory:

```
# crash vmlinux vmcore
```

When `crash` is run on a live system, `/dev/mem` is used as the memory image. In RHEL4, RHEL5 and RHEL6, the `/dev/mem` driver is restricted on x86 and x86_64 systems, and is replaced by the `/dev/crash` driver. In any case, only the kernel object filename is required:

```
# crash vmlinux
```

Furthermore, when `crash` is run on a live system, the `vmlinux` argument is ***not*** required when the kernel object file is located in any of the following locations:

- `/boot`
- `/usr/lib/debug/lib/modules/<release>`
- `/` (root directory)
- any subdirectory of `/usr/src`
- `/usr/src/redhat/BUILD/kernel-x.x.x/linux-<release>`

When the `vmlinux` file is not entered on the command line, a search will be made in all of the

directories above until a kernel object file is found that contains a version string that matches the running kernel, as indicated by /proc/version. If a matching kernel is found, then crash may be invoked on a live system simply by entering:

```
# crash
```

In the examples above, it is presumed that the vmlinux kernel has been built with the -g C flag, which traditionally has not been done by default. To address this requirement, starting with Red Hat Enterprise Linux 3 (RHEL3), all RHEL kernels are now built with -g C flag. The manner of accessing the debug data for RHEL3, RHEL4, RHEL5 and RHEL6 kernels is described in the following sections. Unfortunately, since RHEL2.1 kernels are not built with -g, the kernel must be rebuilt; directions for rebuilding RHEL2.1 kernels can be found here.

RHEL3 Kernels

In RHEL3, the vmlinux kernel debug information is stripped and stored in a separate debuginfo file. The stripped vmlinux file in /boot has an embedded link to its associated debuginfo file in /usr/lib/debug/boot, so that the crash utility (and the built-in gdb module) knows where to find it:

RHEL3 kernel namelist in /boot	RHEL3 kernel debuginfo file in /usr/lib/debug/boot
vmlinux-*<release>*.EL vmlinux-*<release>*.ELsmp vmlinux-*<release>*.ELhugemem	vmlinux-*<release>*.EL.debug vmlinux-*<release>*.ELsmp.debug vmlinux-*<release>*.ELhugemem.debug

The debuginfo files for a specific kernel *<release>* come from a separate RPM that must be installed for the crash utility to work. For example, the i686 RPM for the examples above would be named kernel-debuginfo-*<release>*.i686.rpm, and would install the debuginfo file for all three of the kernel flavors.

For example, to run crash on a live system, the associated debuginfo package must be installed:

```
# uname -r
2.4.21-4.ELsmp
# rpm -ivh kernel-debuginfo-2.4.21-4.EL.i686.rpm
Preparing...                ########################################### [100%]
   1:kernel-debuginfo       ########################################### [100%]
# ls /usr/lib/debug/boot
vmlinux-2.4.21-4.EL.debug
vmlinux-2.4.21-4.ELhugemem.debug
vmlinux-2.4.21-4.ELsmp.debug
#
```

Accordingly, if the running kernel's vmlinux file is in one the search locations above, ***and*** its associated debuginfo file is located in the /usr/lib/debug/boot directory ***or*** in the current directory from which crash is invoked, no arguments are required to run on a live system:

```
# crash
```

However, if the linked debuginfo file is not in either of those locations, it can be added to the crash command line along with the vmlinux filename. So, for example, if the debuginfo file was located in /tmp:

```
# crash /boot/vmlinux-2.4.21-4.ELsmp /tmp/vmlinux-2.4.21-4.ELsmp.debug
```

For analyzing dumpfiles however, the `vmlinux` file name must be on the command line along with the dumpfile name, as in the following examples:

```
# crash /boot/vmlinux-2.4.21-4.ELsmp vmcore
```

or if the debuginfo file is not in the standard location:

```
# crash /boot/vmlinux-2.4.21-4.ELsmp /tmp/vmlinux-2.4.21-4.ELsmp.debug vmcore
```

RHEL4 Kernels

The procedure has been made much simpler for RHEL4 kernels. The kernel is built with the `-g` flag, and the resultant `vmlinux` file is stored in the associated debuginfo package. After installing the debuginfo package, the `vmlinux` file for each kernel flavor of a given RHEL4 release will be installed in the directory named:

`/usr/lib/debug/lib/modules/<`*release*`><`*flavor*`>/vmlinux`

where for i686 kernels, <*flavor*> can be either `hugemem`, `smp`, or nothing (for uniprocessor kernels). For example:

```
# uname -r
2.6.9-6.39.ELsmp
# rpm -ivh kernel-debuginfo-2.6.9-6.39.EL.i686.rpm
Preparing...                ########################################### [100%]
   1:kernel-debuginfo       ########################################### [100%]
#
# find /usr/lib/debug/lib/modules/2.6.9-6.39.EL* -name vmlinux
/usr/lib/debug/lib/modules/2.6.9-6.39.ELhugemem/vmlinux
/usr/lib/debug/lib/modules/2.6.9-6.39.ELsmp/vmlinux
/usr/lib/debug/lib/modules/2.6.9-6.39.EL/vmlinux
#
```

Once the debuginfo package is installed, `crash` can be invoked on the live system with no arguments, because the `vmlinux` file will be found automatically:

```
# crash
```

To run `crash` on a dumpfile, however, the appropriate `vmlinux` file and the dumpfile name must both be on the command line, as in:

```
# crash /usr/lib/debug/lib/modules/2.6.9-6.39.ELsmp/vmlinux vmcore
```

RHEL5 Kernels

RHEL5 kernels are also built with the `-g` flag, and the resultant `vmlinux` file is stored in an associated debuginfo package. Unlike RHEL4, the kernel debuginfo packages are split into one package per flavor plus a "common" package user by all kernel flavors. After installing the debuginfo package, the `vmlinux` file for each kernel flavor of a given RHEL5 release will be installed in the directory named:

`/usr/lib/debug/lib/modules/<`*release*`><`*flavor*`>/vmlinux`

where for i686 kernels, there are 4 possible <*flavor*>s, either `PAE`, `xen`, `debug`, or nothing (for non-PAE SMP kernels). For example:

```
# rpm -ivh kernel-debuginfo-2.6.18-66.el5.i686.rpm \
          kernel-PAE-debuginfo-2.6.18-66.el5.i686.rpm \
          kernel-xen-debuginfo-2.6.18-66.el5.i686.rpm \
          kernel-debuginfo-common-2.6.18-66.el5.i686.rpm
Preparing...                ########################################### [100%]
   1:kernel-debuginfo-common########################################### [ 25%]
   2:kernel-debuginfo       ########################################### [ 50%]
   3:kernel-PAE-debuginfo   ########################################### [ 75%]
   4:kernel-xen-debuginfo   ########################################### [100%]
# find /usr/lib/debug/lib/modules/2.6.18-66.el5* -name vmlinux
/usr/lib/debug/lib/modules/2.6.18-66.el5/vmlinux
/usr/lib/debug/lib/modules/2.6.18-66.el5PAE/vmlinux
/usr/lib/debug/lib/modules/2.6.18-66.el5xen/vmlinux
#
```

Once the debuginfo package is installed, `crash` can be invoked on the live system with no arguments, because the `vmlinux` file will be found automatically:

```
# crash
```

To run `crash` on a dumpfile, however, the appropriate `vmlinux` file and the dumpfile name must both be on the command line, as in:

```
# crash /usr/lib/debug/lib/modules/2.6.18-66.el5/vmlinux vmcore
```

RHEL6 Kernels

RHEL6 kernels are also built with the `-g` flag, and the resultant `vmlinux` file is stored in an associated debuginfo package. Like RHEL5, the kernel debuginfo packages are split into one package per flavor plus a "common" package user by all kernel flavors. After installing the debuginfo package, the `vmlinux` file for each kernel flavor of a given RHEL6 release will be installed in the directory named:

`/usr/lib/debug/lib/modules/<`*release*`><`*flavor*`>/vmlinux`

where for x86_64 kernels, there are only 2 possible <*flavor*>s, either the standard kernel or the `debug` kernel. For example:

```
# rpm -ivh kernel-debuginfo-common-2.6.32-70.el6.x86_64.rpm \
          kernel-debuginfo-2.6.32-70.el6.x86_64.rpm \
          kernel-debug-debuginfo-2.6.32-70.el6.x86_64.rpm
Preparing...                ########################################### [100%]
   1:kernel-debuginfo-common########################################### [ 25%]
   2:kernel-debuginfo       ########################################### [ 50%]
   3:kernel-debug-debuginfo ########################################### [100%]
# find /usr/lib/debug/lib/modules/2.6.32-70.el6* -name vmlinux
/usr/lib/debug/lib/modules/2.6.32-70.el6/vmlinux
/usr/lib/debug/lib/modules/2.6.32-70.el6debug/vmlinux
#
```

Once the debuginfo package is installed, `crash` can be invoked on the live system with no arguments, because the `vmlinux` file will be found automatically:

```
# crash
```

To run `crash` on a dumpfile, however, the appropriate `vmlinux` file and the dumpfile name must both be on the command line, as in:

```
# crash /usr/lib/debug/lib/modules/2.6.32-70.el6/vmlinux vmcore
```

RHEL2.1 Kernels (or kernels built without `-g` flag)

If the running kernel was ***not*** built with the `-g` C flag, then it is necessary to rebuild a kernel of the same configuration with the `-g` C flag. The essential change done by this kernel rebuild task is a modification of top-level `Makefile` of the kernel source tree, such that the `CFLAGS` definition contains the `-g` flag. For example, this is the line that must be changed:

```
CFLAGS := $(CPPFLAGS) -Wall -Wstrict-prototypes -Wno-trigraphs -O2 \
          -fno-strict-aliasing -fno-common
```

by adding the `-g` flag:

```
CFLAGS := $(CPPFLAGS) -Wall -Wstrict-prototypes -Wno-trigraphs -O2 \
          -fno-strict-aliasing -fno-common -g
```

For example, since RHEL2.1 kernels are *not* built with `-g`, a kernel rebuild is required. For a detailed example of how to rebuild a RHEL2.1 kernel with the `-g` flag, please refer to these directions.

Upon rebuilding the kernel, a new `vmlinux` file will be created that contains the debug data required by `crash`. However, the symbol values will not match those of the running or dumped kernel. To deal with this inequity, the actual symbol values can be gathered from either the original *non*-debug `vmlinux` file *or* from its associated System.map file. That being the case, two arguments must be supplied to `crash` to fully describe the running/dumped kernel, the newly-created `vmlinux` file compiled with `-g`, as well as a source of the real symbol values. So, for example, if the `vmlinux` file built with `-g` were renamed to `vmlinux.dbg`, the invocation line would look like this on a live system:

```
# crash vmlinux vmlinux.dbg
(or)
# crash /boot/System.map vmlinux.dbg
(or)
# crash -S vmlinux.debug
```

The `-S` argument above is simply an alternative to entering the default `/boot/System.map` string.

Similarly, when looking at a dumpfile, two arguments are required to describe the dumped kernel, along with the `vmcore` image:

```
# crash vmlinux vmlinux.dbg vmcore
(or)
# crash /boot/System.map vmlinux.dbg vmcore
(or)
# crash -S vmlinux.dbg vmcore
```

Again, for a detailed example of how to rebuild a RHEL2.1 kernel with the `-g` flag, refer to these directions.

Invocation output

The arguments may be entered in any order. If the file arguments are not in the current directory, absolute pathnames must be used. When in doubt, simply enter `crash -h` to get an explanation of the command line arguments:

```
# crash -h

Usage:
  crash [-h [opt]][-v][-s][-i file][-d num] [-S] [mapfile] [namelist] [dumpfile]
```

```
  [namelist]
    The [namelist] argument is a pathname to an uncompressed kernel image
    (a vmlinux file) that has been compiled with the "-g" switch, or
    that has an accessible, associated, debuginfo file.  If the [dumpfile]
    argument is entered, then the [namelist] argument must be entered
    If the [namelist] argument is not entered when running on a live
    system, a search will be made in several typical directories for
    for a kernel namelist file that matches the live system.

  [dumpfile]
    The [dumpfile] argument is a pathname to a kernel memory core dump
    file.  If the [dumpfile] argument is not entered, the session will be
    invoked on the live system using /dev/mem, which usually requires root
    privileges.

  [mapfile]
    If the live system kernel, or the kernel from which the [dumpfile]
    was derived, was not compiled with the -g switch, then the additional
    [mapfile] argument is required.  The [mapfile] argument may consist
    of either the associated System.map file, or the non-debug kernel
    namelist.  However, if the [mapfile] argument is used, then the
    [namelist] argument must be a kernel namelist of a similar kernel
    version that was built with the -g switch.

  [-S]
    Use "/boot/System.map" as the [mapfile].

    Examples when running on a live system:

      $ crash
      $ crash /usr/tmp/vmlinux
      $ crash /boot/System.map vmlinux.dbg
      $ crash -S vmlinux.dbg
      $ crash vmlinux vmlinux.dbg

    Examples when running on a dumpfile:

      $ crash vmlinux vmcore
      $ crash /boot/System.map vmlinux.dbg vmcore
      $ crash -S vmlinux.dbg vmcore
      $ crash vmlinux vmlinux.dbg vmcore

  [-h [opt]]
    The -h option alone displays this message.  If the [opt] argument is
    a crash command name, the help page for that command is displayed.  If
    the string "input" is entered, a page describing the various crash
    command line input options is displayed.  If the string "output" is
    entered, a page describing command line output options is displayed.

  [-v]
    Display the versions of crash and gdb making up this executable.

  [-s]
    Do not display any version, GPL, or crash initialization data; proceed
    directly to the "crash>" prompt.

  [-i file]
    Execute the crash command(s) in [file] prior to accepting any user
    input from the "crash>" prompt.

  [-d num]
    Set crash debug level [num].  The higher the number, the more debug data
    will be printed during crash runtime.
```

Given that all invocation arguments are in order, here is an example of a successful invocation on a dumpfile, running a kernel that was built with `-g`, along with a `vmcore` dump file was created by the Red Hat Netdump facility:

```
# crash vmlinux-2.4.20-2.1.15.entsmp vmcore

crash 4.0-8.11
Copyright (C) 2002, 2003, 2004, 2005, 2006, 2007, 2008  Red Hat, Inc.
```

```
Copyright (C) 2004, 2005, 2006  IBM Corporation
Copyright (C) 1999-2006  Hewlett-Packard Co
Copyright (C) 2005, 2006  Fujitsu Limited
Copyright (C) 2006, 2007  VA Linux Systems Japan K.K.
Copyright (C) 2005  NEC Corporation
Copyright (C) 1999, 2002, 2007  Silicon Graphics, Inc.
Copyright (C) 1999, 2000, 2001, 2002  Mission Critical Linux, Inc.
This program is free software, covered by the GNU General Public License,
and you are welcome to change it and/or distribute copies of it under
certain conditions.  Enter "help copying" to see the conditions.
This program has absolutely no warranty.  Enter "help warranty" for details.

GNU gdb 6.1
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i686-pc-linux-gnu"...

      KERNEL: vmlinux-2.4.20-2.1.15.entsmp
    DUMPFILE: vmcore
        CPUS: 1
        DATE: Wed Mar 12 10:12:56 2003
      UPTIME: 00:38:25
LOAD AVERAGE: 1.16, 0.74, 0.30
       TASKS: 60
    NODENAME: dhcp64-220.boston.redhat.com
     RELEASE: 2.4.20-2.1.15.entsmp
     VERSION: #1 SMP Tue Mar 11 16:12:22 EST 2003
     MACHINE: i686  (501 Mhz)
      MEMORY: 128 MB
       PANIC: "Oops: 0002" (check log for details)
         PID: 0
     COMMAND: "swapper"
        TASK: c038e000
         CPU: 0
       STATE: TASK_RUNNING (PANIC)

crash>
```

This next example shows the output when the panicking kernel was ***not*** built with `-g`. In this case, a similar kernel type was built with `-g`, and the resultant kernel object file was renamed as `vmlinux.dbg`. Note that there will be a message concerning the patching of `gdb` data; this indicates that the non-matching symbol values from the `vmlinux.dbg` are being over-written by the correct symbol values found in the original `vmlinux` file:

```
# crash vmlinux vmlinux.dbg vmcore

crash 4.0-8.11
Copyright (C) 2002, 2003, 2004, 2005, 2006, 2007, 2008  Red Hat, Inc.
Copyright (C) 2004, 2005, 2006  IBM Corporation
Copyright (C) 1999-2006  Hewlett-Packard Co
Copyright (C) 2005, 2006  Fujitsu Limited
Copyright (C) 2006, 2007  VA Linux Systems Japan K.K.
Copyright (C) 2005  NEC Corporation
Copyright (C) 1999, 2002, 2007  Silicon Graphics, Inc.
Copyright (C) 1999, 2000, 2001, 2002  Mission Critical Linux, Inc.
This program is free software, covered by the GNU General Public License,
and you are welcome to change it and/or distribute copies of it under
certain conditions.  Enter "help copying" to see the conditions.
This program has absolutely no warranty.  Enter "help warranty" for details.

GNU gdb 6.1
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i686-pc-linux-gnu"...

please wait... (patching 16053 gdb minimal_symbol values)
```

```
DEBUG KERNEL: vmlinux.dbg
    DUMPFILE: vmcore
        CPUS: 1
        DATE: Wed Mar 27 11:02:31 2002
      UPTIME: 00:07:24
LOAD AVERAGE: 0.43, 0.42, 0.19
       TASKS: 68
    NODENAME: anderson.boston.redhat.com
     RELEASE: 2.4.9-26beta.48enterprise
     VERSION: #1 SMP Thu Mar 21 12:33:05 EST 2002
     MACHINE: i686  (501 Mhz)
      MEMORY: 128 MB
       PANIC: "Oops: 0002" (check log for details)
         PID: 1696
     COMMAND: "insmod"
        TASK: c74de000
         CPU: 0
       STATE: TASK_RUNNING (PANIC)

crash>
```

Invocation on a live system looks essentially the same, except that the DUMPFILE will be indicated as /dev/mem. In the following example, no arguments were entered, because the running RHEL3 kernel was found in the /boot directory, and its associated debuginfo file in the /usr/lib/debug/boot directory. The debuginfo file is listed next to the DEBUGINFO tag:

```
# crash

crash 4.0-8.11
Copyright (C) 2002, 2003, 2004, 2005, 2006, 2007, 2008  Red Hat, Inc.
Copyright (C) 2004, 2005, 2006  IBM Corporation
Copyright (C) 1999-2006  Hewlett-Packard Co
Copyright (C) 2005, 2006  Fujitsu Limited
Copyright (C) 2006, 2007  VA Linux Systems Japan K.K.
Copyright (C) 2005  NEC Corporation
Copyright (C) 1999, 2002, 2007  Silicon Graphics, Inc.
Copyright (C) 1999, 2000, 2001, 2002  Mission Critical Linux, Inc.
This program is free software, covered by the GNU General Public License,
and you are welcome to change it and/or distribute copies of it under
certain conditions.  Enter "help copying" to see the conditions.
This program has absolutely no warranty.  Enter "help warranty" for details.

GNU gdb 6.1
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i686-pc-linux-gnu"...

      KERNEL: /boot/vmlinux-2.4.21-4.ELhugemem
   DEBUGINFO: /usr/lib/debug/boot/vmlinux-2.4.21-4.ELhugemem.debug
    DUMPFILE: /dev/mem
        CPUS: 2
        DATE: Thu Aug 21 11:24:38 2003
      UPTIME: 1 days, 23:14:11
LOAD AVERAGE: 0.14, 0.10, 0.08
       TASKS: 106
    NODENAME: crash.boston.redhat.com
     RELEASE: 2.4.21-4.ELhugemem
     VERSION: #1 SMP Wed Aug 13 21:33:31 EDT 2003
     MACHINE: i686  (1993 Mhz)
      MEMORY: 511.5 MB
         PID: 4757
     COMMAND: "crash"
        TASK: 19b82000
         CPU: 1
       STATE: TASK_RUNNING (ACTIVE)

crash>
```

Invocation Errors

Invocation errors will cause the `crash` session to abort upon initialization. Typically they occur as the result of one of the following reasons:

1. The `vmlinux` file contains no debug data (i.e., was built without the `-g` flag), and no additional debug kernel object file name was entered on the command line. The error message will be of the form:

   ```
   crash: /boot/vmlinux-2.4.18-14: no debugging data available
   ```

2. The `vmlinux` file does not match the dumpfile. The error message will be of the form:

   ```
   crash: vmlinux and tmp/vmcore do not match!
   ```

3. The `vmlinux` file could not be found on a live system. The error message will be of the form:

   ```
   crash: cannot find booted kernel -- please enter namelist argument
   ```

4. The associated debuginfo file cannot be found. The error message will be of the form:

   ```
   crash: /boot/vmlinux-2.4.21-4.ELsmp: no debugging data available
   crash: vmlinux-2.4.21-4.ELsmp.debug: debuginfo file not found
   ```

5. The `crash` utility binary does not match the `vmlinux` and/or `vmcore` arguments. The error message will be of the form:

   ```
   WARNING: machine type mismatch:
            crash utility: X86
            vmlinux: X86_64

   crash: vmlinux: not a supported file format
   ```

Command Input

Upon a successful session invocation on a dump file or a live kernel, the `crash>` prompt will appear. Interactive `crash` commands are gathered using the GNU `readline` library, taking advantage of its command line history mechanism, and its `vi` or `emacs` command line editing modes. Commands may also be issued to `crash` from a file.

Command Line History

The command line history consists of a numbered list of previously-run commands. The full list of commands may be viewed by entering `h` at any time. For example:

```
crash> h

[1] bt -a
[2] ps
[3] foreach bt
[4] set
[5] dis -rl c0221141
```

```
crash>
```

Commands in the history list may be re-run in the following manners

1. To re-run the ***last*** command executed, simply enter `r` or `!!` and then ENTER.
2. Enter `r` followed by the appropriate history list number, and then ENTER.
3. Enter `r` followed by a uniquely-identifying set of characters from the beginning of the previously-entered command string, and then ENTER.
4. Recycle back through the command history list using the up-arrow and down-arrow keys until the desired command is re-displayed, and then ENTER.
5. Recycle back through the command history list using the key-strokes appropriate for the command line editing mode being used (`vi` or `emacs`) until the desired command is re-displayed, and then ENTER.

Command Line Editing

The command line editing mode may be set to either `vi` (the default) or `emacs`. The mode may set in the following manners, listed in increasing order of precedence:

1. Do none of the following, in which case the default `vi` editing mode will be used.
2. Set the `EDITOR` environment variable to either `vi` or `emacs`.
3. Create an entry in a `.crashrc` file in the user's `$HOME` directory. The entry must be a line of one of the following forms:

   ```
   set vi
   set emacs
   ```

4. Create an entry in a `.crashrc` file be located in the current directory, in the form shown in (3) above.
5. Use the `-e` command line option, as in:

   ```
   # crash -e [vi | emacs] ...
   ```

Given either editing mode, any previously entered command line can be brought back by entering the mode-specific key-stroke(s), the command line edited using the proper mode, and then run by hitting ENTER.

Command Line Input from a File

An input file consisting of a list of commands may be fed to `crash` in the following manners:

1. Upon invocation, as in:

   ```
   # crash -i inputfile
   ```

2. Upon invocation, by entering the commands in a `.crashrc` file, which can be either in the user's `$HOME` directory or in the current directory.
3. On the command line during a `crash` session, as in:

   ```
   crash> < inputfile
   ```

In all of the three cases above, after the list of commands in the file have completed, the `crash>` prompt will appear and commands may then be entered interactively (unless one of the file commands happens to be the `exit` command).

Numerical Arguments

Numerical arguments are typically presumed to be decimal unless the argument contains an `a`, `b`, `c`, `d`, `e` or `f`. In those cases, the preceding `0x` is ***not*** required. For hexadecimal numbers that do not contain one of those 6 characters, the preceding `0x` is required. So, for example, a value of 1 gigabyte would have to be expressed as `0x40000000`, whereas 3 gigabytes could be expressed as `c0000000`.

It should be noted that several commands will ***only*** accept hexadecimal numerical arguments. For example, the `rd` ("read") command only accepts hexadecimal addresses. Therefore a read from user address of 0x40017000 could be entered as:

```
crash> rd 40017000 40
40017000:  20000824 00000010 00000048 00000063   $.. ....H...c...
40017010:  00000082 000000ba 000000bb 000000cd   ................
40017020:  000000ce 000000cf 000000d7 000000db   ................
40017030:  000000dc 000000dd 000000de 000000e2   ................
40017040:  000000ed 00000167 6c676e45 20687369   ....g...English
40017050:  61636f6c 6620656c 7420726f 55206568   locale for the U
40017060:  46004153 20656572 74666f53 65726177   SA.Free Software
40017070:  756f4620 7461646e 2c6e6f69 636e4920    Foundation, Inc
40017080:  3935002e 6d655420 20656c70 63616c50   ..59 Temple Plac
40017090:  202d2065 74697553 33332065 42202c30   e - Suite 330, B
```

Command Output

`crash` commands can often be verbose, and it's helpful to control the output, as well as to be able to scroll backwards to view previous command output. So, by default, command output that would overflow the user's display screen is piped to `/usr/bin/less`, along with a prompt line that informs the user how to scroll forward, backward, or to quit the command. For example, here is an example of what a `ps` command might look like:

```
crash> ps
   PID    PPID  CPU   TASK    ST  %MEM   VSZ   RSS  COMM
     0      0   0  c030a000  RU   0.0     0     0  [swapper]
     1      0   0  cff98000  IN   0.2  1412   468  init
     2      1   0  c1446000  IN   0.0     0     0  [keventd]
     3      1   0  cfffa000  IN   0.0     0     0  [kapm-idled]
     4      0   0  cfff8000  IN   0.0     0     0  [ksoftirqd_CPU0]
     5      0   0  cffee000  IN   0.0     0     0  [kswapd]
     6      0   0  cffec000  IN   0.0     0     0  [kreclaimd]
     7      0   0  c1826000  IN   0.0     0     0  [bdflush]
     8      0   0  c1824000  IN   0.0     0     0  [kupdated]
     9      1   0  cff90000  IN   0.0     0     0  [mdrecoveryd]
    13      1   0  cf07a000  IN   0.0     0     0  [kjournald]
    89      1   0  ce804000  IN   0.0     0     0  [khubd]
   184      1   0  ce4d4000  IN   0.0     0     0  [kjournald]
   572      1   0  cd938000  IN   0.0   440    48  dhcpcd
   637      1   0  ce4a4000  IN   0.2  1476   612  syslogd
   642      1   0  cd92c000  IN   0.2  2092   432  klogd
   663      1   0  ce2bc000  IN   0.2  1564   612  portmap
   691      1   0  cd84a000  IN   0.3  1652   668  rpc.statd
   803      1   0  cd756000  IN   0.2  1400   452  apmd
   828      1   0  cd6c2000  IN   0.3 18024   684  ypbind
   830    828   0  cd76e000  IN   0.3 18024   684  ypbind
   831    830   0  cd71c000  IN   0.3 18024   684  ypbind
-- MORE --  forward: <SPACE>, <ENTER> or j  backward: b or k  quit: q
```

This default output scrolling behavior can be turned off by entering the following line in a `.crashrc` file located in either the `$HOME` or current directories:

```
set scroll off
```

During runtime, the following commands (or their respective builtin aliases) can be used to turn the scrolling behavior off, and back on, again:

```
crash> set scroll off
scroll: off
crash> set scroll on
scroll: on
crash> alias

ORIGIN   ALIAS    COMMAND
builtin  man      help
builtin  ?        help
builtin  quit     q
builtin  sf       set scroll off
builtin  sn       set scroll on
builtin  hex      set radix 16
builtin  dec      set radix 10
builtin  g        gdb
builtin  px       p -x
builtin  pd       p -d
builtin  for      foreach
builtin  size     *
builtin  dmesg    log
builtin  last     ps -l
crash> sf
scroll: off
crash> sn
scroll: on
crash>
```

Alternatively, command output may be redirected to a pipe or to a file using standard shell redirection syntax. For examples:

```
crash> task | grep uid
  uid = 3369,
  euid = 3369,
  suid = 3369,
  fsuid = 3369,
crash> foreach bt > bt.all
crash> ps >> process.data
crash> kmem -i | grep SLAB > slab.pages
crash>
```

When a command's output is redirected to a pipe or file, the default `/usr/bin/less` behavior is turned off for that particular command.

Numerical Output

The default numerical output radix for non-pointer values is decimal, which is most often noticed when using the builtin `gdb` capability of printing formatted data structures. During runtime, the following commands (or their respective builtin aliases) can be used to toggle the output radix from decimal to hexadecimal, and back again:

```
crash> set radix 16
output radix: 16 (hex)
crash> set scroll 10
output radix: 10 (decimal)
crash> alias

ORIGIN   ALIAS    COMMAND
builtin  man      help
builtin  ?        help
builtin  quit     q
builtin  sf       set scroll off
```

```
builtin  sn       set scroll on
builtin  hex      set radix 16
builtin  dec      set radix 10
builtin  g        gdb
builtin  px       p -x
builtin  pd       p -d
builtin  for      foreach
builtin  size     *
builtin  dmesg    log
crash> hex
output radix: 16 (hex)
crash> dec
output radix: 10 (decimal)
crash>
```

Alternatively, the `px` or `pd` aliases coerce the "print" command `p`, to override the current output radix. For example, here the changing value of `jiffies` on a live system is printed using the current default radix, then in hexadecimal, and lastly in decimal:

```
crash> p jiffies
jiffies = $4 = 69821055
crash> px jiffies
jiffies = $5 = 0x42963aa
crash> pd jiffies
jiffies = $6 = 69821656
crash>
```

Crash Context

Upon a successful invocation of a `crash` session, one of the existing Linux tasks is selected as the ***current context***. It is important to be aware of the current context because several `crash` commands are "context-sensitive", meaning that the command is executed from the view-point of the current context. Therefore, the output of context-sensitive commands can vary depending upon which context is current.

Upon invocation of a `crash` session, the selection of the current context is based upon the following criteria:

On dumpfiles:

- The task that was running when `die()` was called.
- The task that was running when `panic()` was called.
- The task that was running when an ALT-SYSRQ-c keyboard interrupt was received.
- The task that was running when the character "c" was echoed to `/proc/sysrq-trigger`.

On a live system:

- the `crash` task itself.

The current context selection is shown in the session invocation data. For example, here is a session begun on a dumpfile that was created when an `insmod` task's attempt to install a module resulted in an "oops" violation:

```
# crash tmp/vm*

crash 4.0-8.11
```

```
Copyright (C) 2002, 2003, 2004, 2005, 2006, 2007, 2008  Red Hat, Inc.
Copyright (C) 2004, 2005, 2006  IBM Corporation
Copyright (C) 1999-2006  Hewlett-Packard Co
Copyright (C) 2005, 2006  Fujitsu Limited
Copyright (C) 2006, 2007  VA Linux Systems Japan K.K.
Copyright (C) 2005  NEC Corporation
Copyright (C) 1999, 2002, 2007  Silicon Graphics, Inc.
Copyright (C) 1999, 2000, 2001, 2002  Mission Critical Linux, Inc.
This program is free software, covered by the GNU General Public License,
and you are welcome to change it and/or distribute copies of it under
certain conditions.  Enter "help copying" to see the conditions.
This program has absolutely no warranty.  Enter "help warranty" for details.

GNU gdb 6.1
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i686-pc-linux-gnu"...

      KERNEL: tmp/vmlinux
DEBUG KERNEL: tmp/vmlinux.dbg
    DUMPFILE: tmp/vmcore
        CPUS: 1
        DATE: Wed Mar 27 11:02:31 2002
      UPTIME: 00:07:24
LOAD AVERAGE: 0.43, 0.42, 0.19
       TASKS: 68
    NODENAME: anderson.boston.redhat.com
     RELEASE: 2.4.9-26beta.48enterprise
     VERSION: #1 SMP Thu Mar 21 12:33:05 EST 2002
     MACHINE: i686  (501 Mhz)
      MEMORY: 128 MB
       PANIC: "Oops: 0002" (check log for details)
         PID: 1696
     COMMAND: "insmod"
        TASK: c74de000
         CPU: 0
       STATE: TASK_RUNNING (PANIC)

crash>
```

During runtime, the current context can always be displayed by entering the set command with no arguments:

```
crash> set
    PID: 1696
COMMAND: "insmod"
   TASK: c74de000
    CPU: 0
  STATE: TASK_RUNNING (PANIC)
crash>
```

Changing the Crash Context

The current context can be changed to a new task via the set command. Either of two "handles" may be used to identify a task, the PID number, or the kernel address of the task's task_struct. For example:

```
crash> set 1
    PID: 1
COMMAND: "init"
   TASK: c7f98000
    CPU: 0
  STATE: TASK_RUNNING
crash> set c0a52000

    PID: 1503
COMMAND: "cat"
```

```
   TASK: c0a52000
    CPU: 0
  STATE: TASK_INTERRUPTIBLE
crash>
```

Alternatively, the current context can be set to the task running on a given CPU number, or back to the panicking task. Using the same dumpfile session shown above, in which there is only one CPU, the original context may be restored using the `-c` *`CPU-number`* or the `-p` ("panic task") options:

```
crash> set -c 0
    PID: 1696
COMMAND: "insmod"
   TASK: c74de000
    CPU: 0
  STATE: TASK_RUNNING (PANIC)
crash> set -p
    PID: 1696
COMMAND: "insmod"
   TASK: c74de000
    CPU: 0
  STATE: TASK_RUNNING (PANIC)
crash>
```

Context-Sensitive Commands

It is important to be aware that several `crash` commands are context-sensitive. For example, the `files` command displays data about the open files of a task. If it is issued with no arguments, it displays the open files data of the current context. In this example, the current context happens to be PID 642, the `klogd` daemon:

```
crash> files

PID: 642    TASK: cd92c000  CPU: 0   COMMAND: "klogd"
ROOT: /    CWD: /
 FD    FILE     DENTRY    INODE    TYPE  PATH
  0  ce06c800  ce29ec60  cd8df900  REG   /proc/kmsg
  1  ce06cf20  ce29ebe0  cd8df740  SOCK  socket:/[858]
  2  ce06c5c0  ce2423a0  ce462c80  REG   /boot/System.map-2.4.9-e.3enterprise
```

However, if the `files` command is issued with either of the two task handles as an argument, then it will display the open files data of the specified task. In this example, PID 12731 is specified:

```
crash> files 12731

PID: 12731  TASK: c8150000  CPU: 0   COMMAND: "vi"
ROOT: /    CWD: /tmp
 FD    FILE     DENTRY    INODE    TYPE  PATH
  0  c988cd80  ced919a0  c87fc3c0  CHR   /dev/pts/11
  1  c988cd80  ced919a0  c87fc3c0  CHR   /dev/pts/11
  2  c988cd80  ced919a0  c87fc3c0  CHR   /dev/pts/11
  4  c2927ae0  c6cad8a0  cd6d5040  REG   /tmp/.crontab.12730.swp
  5  c2927a80  c6cad9a0  c5764ac0  REG   /tmp/crontab.12730
```

This type of context-sensitive behaviour is also exhibited by the `vm`, `bt`, `sig`, `set`, `net` and `task` commands. Unless a PID or task address is specified as an argument, the output will reflect data concerning the current context.

Other commands may simply default to the current context. For example, the `rd` command can read memory from an address that is specified as a user-space address. Since the `rd` command does not accept a PID or task address as an argument, it would be necessary to be aware that the user-space access will come from the address space of the current

context.

Builtin Help

Readily available help information is built into the `crash` utility. During a session, entering the `help` command with no argument shows the following menu:

```
crash> help

*               files           mod             runq            union
alias           foreach         mount           search          vm
ascii           fuser           net             set             vtop
bt              gdb             p               sig             waitq
btop            help            ps              struct          whatis
dev             irq             pte             swap            wr
dis             kmem            ptob            sym             q
eval            list            ptov            sys
exit            log             rd              task
extend          mach            repeat          timer

crash version: 4.0-8.11   gdb version: 6.1
For help on any command above, enter "help <command>".
For help on input options, enter "help input".
For help on output options, enter "help output".

crash>
```

Each command has its own `man`-like help page, which can be viewed by clicking on the command name above. Each help page details the syntax of the command and its available options, a description of the command in general, a description of each option, and a set of examples. During a `crash` session, a command's help page can be displayed by entering `help` followed by the command name. So, for example, to get help on how to use the `set` command:

```
crash> help set

NAME
  set - set a process context or internal crash variable

SYNOPSIS
  set [pid | taskp | [-c cpu] | -p] | [crash_variable [setting]] | -v

DESCRIPTION
  This command either sets a new context, or gets the current context for
  display.  The context can be set by the use of:

      pid  a process PID.
    taskp  a hexadecimal task_struct pointer.
   -c cpu  sets the context to the active task on a cpu (dumpfiles only).
       -p  sets the context to the panic task, or back to the crash task on
           a live system.
       -v  display the current state of internal crash variables.

  If no argument is entered, the current context is displayed.  The context
  consists of the PID, the task pointer, the CPU, and task state.

  This command may also be used to set internal crash variables.  If no value
  argument is entered, the current value of the crash variable is shown.  These
  are the crash variables, acceptable arguments, and purpose:
```

```
        scroll  on | off     controls output scrolling.
        scroll  less         /usr/bin/less as the output scrolling program.
        scroll  more         /bin/more as the output scrolling program.
        scroll  CRASHPAGER   use CRASHPAGER environment variable as the
                             output scrolling program.
         radix  10 | 16      sets output radix to 10 or 16.
       refresh  on | off     controls internal task list refresh.
     print_max  number       set maximum number of array elements to print.
       console  device-name  sets debug console device.
         debug  number       sets crash debug level.
          core  on | off     if on, drops core when the next error message
                             is displayed.
          hash  on | off     controls internal list verification.
        silent  on | off     turns off initialization messages; turns off
                             crash prompt during input file execution.
                             (scrolling is turned off if silent is on)
          edit  vi | emacs   set line editing mode (from .crashrc file only).
      namelist  filename     name of kernel (from .crashrc file only).
      dumpfile  filename     name of core dumpfile (from .crashrc file only).
 zero_excluded  on | off     controls whether excluded pages from a dumpfile
                             should return zero-filled memory.

  Internal variables may be set in four manners:

    1. entering the set command in $HOME/.crashrc.
    2. entering the set command in .crashrc in the current directory.
    3. executing an input file containing the set command.
    4. during runtime with this command.

  During initialization, $HOME/.crashrc is read first, followed by the
  .crashrc file in the current directory.  Set commands in the .crashrc file
  in the current directory override those in $HOME/.crashrc.  Set commands
  entered with this command or by runtime input file override those
  defined in either .crashrc file.  Multiple set command arguments or argument
  pairs may be entered in one command line.

EXAMPLES
  Set the current context to task c2fe8000:

    crash> set c2fe8000
        PID: 15917
    COMMAND: "bash"
       TASK: c2fe8000
        CPU: 0
      STATE: TASK_INTERRUPTIBLE

  Set the context back to the panicking task:

    crash> set -p
        PID: 698
    COMMAND: "gen12"
       TASK: f9d78000
        CPU: 2
      STATE: TASK_RUNNING (PANIC)

  Turn off output scrolling:

    crash> set scroll off
    scroll: off (/usr/bin/less)

  Show the current state of crash internal variables:

    crash> set -v
            scroll: on (/usr/bin/less)
             radix: 10 (decimal)
           refresh: on
         print_max: 256
           console: /dev/pts/2
             debug: 0
              core: off
              hash: on
            silent: off
              edit: vi
          namelist: vmlinux
```

```
          dumpfile: vmcore
      zero_excluded: off

  Show the current context:

    crash> set
         PID: 1525
     COMMAND: "bash"
        TASK: c1ede000
         CPU: 0
       STATE: TASK_INTERRUPTIBLE
```

If for some reason a `crash` session cannot be invoked, but help information for a particular `crash` command is desired, the same help page can be displayed from a shell command line using the `-h` option to `crash`:

```
# crash -h ascii

NAME
  ascii - translate a hexadecimal string to ASCII

SYNOPSIS
  ascii value ...

DESCRIPTION
  Translates 32-bit or 64-bit hexadecimal values to ASCII.  If no argument
  is entered, an ASCII chart is displayed.

EXAMPLES
  Translate the hexadecimal value of 0x62696c2f7273752f to ASCII:

    crash> ascii 62696c2f7273752f
    62696c2f7273752f: /usr/lib

  Display an ASCII chart:

    crash> ascii

         0    1   2   3   4   5   6   7
       +--------------------------------
     0 | NUL DLE  SP  0   @   P   '   p
     1 | SOH DC1  !   1   A   Q   a   q
     2 | STX DC2  "   2   B   R   b   r
     3 | ETX DC3  #   3   C   S   c   s
     4 | EOT DC4  $   4   D   T   d   t
     5 | ENQ NAK  %   5   E   U   e   u
     6 | ACK SYN  &   6   F   V   f   v
     7 | BEL ETB  `   7   G   W   g   w
     8 |  BS CAN  (   8   H   X   h   x
     9 |  HT  EM  )   9   I   Y   i   y
     A |  LF SUB  *   :   J   Z   j   z
     B |  VT ESC  +   ;   K   [   k   {
     C |  FF  FS  ,   <   L   \   l   |
     D |  CR  GS  _   =   M   ]   m   }
     E |  SO  RS  .   >   N   ^   n   ~
     F |  SI  US  /   ?   O   -   o  DEL

#
```

Lastly, help concerning command input and output can be displayed by entering **`help input`** or **`help output`** during runtime, or `crash -h input` or `crash -h output` from a shell command line.

The Command Set

Each `crash` command generally falls into one of the following categories:

- **Symbolic display of kernel text or data**
- **System state**
- **Utility functions**
- **Session Control Commands**

The remainder of this section breaks the command set into categories, and gives a short description of each command in that category. However, for complete details and examples, recall that the `crash` utility has a self-contained help page for each command; to view the full help page, click on the command name next to its description below.

Symbolic Display of Kernel Text or Data

The following commands typically take full advantage of the power of `gdb` to display kernel data structures symbolically.

Command	Description
`struct`	Displays a formatted kernel data structure type located at a given address, or at an address referred to by a symbol; if no address is specified, the structure definition is displayed. The output can be narrowed down to a singular member of the structure, or to display the offset of every member from the beginning of the structure. A count may be appended to display an array of structures. Its usage is so common that two short-cuts exist such that the user need not enter the "`struct`" command name: 1. The "pointer-to" `*` command below can be substituted. 2. If a structure name is entered as the first token on a command line, the "`struct`" command is actually not necessary.
`union`	Same as `struct` command, but used for kernel data types defined as unions instead of structures..
`*`	"Pointer-to" command which can be used in lieu of entering `struct` or `union`; the `gdb` module first determines whether the argument is a structure or a union, and then calls the appropriate function.
`p`	Displays the contents of a kernel variable; the arguments are passed on to `gdb`'s `print` command for proper formatting. Two builtin aliases, `px` and `pd`, set the numerical output radix to hexadecimal or decimal for the print operation, temporarily overriding the current default.
`whatis`	Displays all available symbol table information concerning a data type or a data symbol.
`sym`	Translates a kernel symbol name to its kernel virtual address and section, or a kernel virtual address to its symbol name and section. It can also be used to dump the complete list of kernel symbols, or to query the symbol list for all symbols containing a given sub-string.
`dis`	Disassembles the text of complete kernel function, or from a specified address for a given number of instructions, or from the beginning of a function up to a specified address.

System State

The majority of `crash` commands come from the following set of "kernel-aware" commands, which delve into various kernel subsystems on a system-wide or per-task basis. The task-specific commands are context-sensitive, meaning that they act upon the current context unless a PID or task address is specified as an argument.

Command	Description
bt	Arguably the most useful `crash` command, `bt` displays a task's kernel stack back-trace, including full exception frame dumps. It is context-sensitive, although the `-a` option will display the stack traces of the active task on each CPU. This command is often used within the foreach wrapper command in order to display the back traces of all tasks with one command.
dev	Displays data concerning the character and block device assignments, I/O port usage, I/O memory usage, and PCI device data.
files	This context-sensitive command displays the task's current root directory and working directories, and then for each open file descriptor, shows: ◦ its `file` struct address ◦ its `dentry` struct address ◦ its `inode` struct address ◦ the file type ◦ the file's full pathname Another option acts upon a specified `dentry` address, showing: ◦ its `inode` struct address ◦ its `superblock` struct address ◦ the file type ◦ the file's full pathname It can be called from the foreach wrapper command.
fuser	Displays a list of tasks that reference a specified filename or `inode` address as the current root or working directory, an open file descriptor, or which mmap the file.
irq	Display data concerning interrupt request numbers and bottom-half handling.
kmem	This command has numerous options that delve into the state of several kernel memory subsystems: ◦ general memory usage, similar in scope to `/proc/meminfo` ◦ `kmalloc` slab memory allocator, including an option that lists each slab object and its state, verifying the slab chain ◦ display and verification of free page lists ◦ `vmalloc` memory allocator `vmlist` contents ◦ display and verification of the page cache ◦ the `mem_map` page list

- display NUMA data, if applicable

Also, given an address, this command searches the symbol table, the slab subsystem, the free list, the `page_hash_table`, the `vmlist`, and the `mem_map` array, displaying where it was found.

log Dumps the kernel message buffer chronologically, accounting for any wrap-around.

mach Displays machine and/or processor specific data.

mod Displays the list of currently-loaded kernel modules. More importantly, it loads the debug data from the module object files if they are available, allowing symbolic debugging capability of kernel modules.

mount For each mounted filesystem, or for just a specified filesystem, displays:

- its `vfsmount` struct address
- its `super_block` struct address
- its type
- its device name
- its mount point

Options exist to dump a list of a specified filesystem's open files or dirty inodes. Filesystems may be specified by `vfsmount`, `super_block`, or `inode` addresses, or by device name or mount point names.

net Displays various network-related data:

- displays each configured network device's `net_device` address, its name, and IP address
- displays the ARP cache
- context-sensitive display of information concerning the open sockets of a task
- translates an IP address expressed as a decimal or hexadecimal value into a standard numbers-and-dots notation

It can be called from the `foreach` wrapper command.

ps Useful process status command, in typical Linux `ps` command type output, containing:

- PID number
- PPID number
- CPU number
- task address
- process state
- percent of physical memory consumed
- virtual address size
- resident set size
- command name

Also has an option to show a task's parental hierarchy back to the `init` process, and another to show all children of a task.

pte This command translates the contents of a PTE into its physical page address and page bit settings, or if it references a swap location, the

swap device and offset.

runq Displays list of tasks on the run queue.

sig A context-sensitive command which displays a task's signal information, including:

- whether an unblocked signal is pending
- the pending and blocked signals
- the handler data for each signal
- queued signals, if any

Other options list the signal number/names combination for a processor type, and translate the contents of a `sigset_t` into the signal names whose bits are set. It can be called from the foreach wrapper command.

swap For each configured swap device, this command displays the same data that is shown by the Linux command `swapon -s`.

sys Re-displays the same system-related data that is seen during `crash` initialization:

- the kernel object filename
- the dumpfile name
- the number of CPUS
- the date
- system uptime
- system load average
- the number of tasks
- the nodename
- the kernel release and version data
- the processor type and speed
- the amount of memory
- the panic string

Other options display information concerning the system call table, and one allows the `root` user to panic a live system.

task This context-sensitive command displays a task's complete `task_struct` contents, or one or more members of the structure. This command is often used within the foreach wrapper command in order to display `task_struct` data for all tasks with one command.

timer Displays the timer queue entries in chronological order, listing the target function names, the current value of `jiffies`, and the expiration time of each entry.

vm This powerful, context-sensitive command displays a wealth of information concerning a task's virtual memory data, including:

- its `mm_struct` address
- its page directory address
- its resident set size
- its total virtual memory size
- each `vm_area_struct` address, along with its start and ending virtual address, flags, and source file if applicable.
- optionally, every virtual page referenced by a `vm_area_struct` can be

translated into its physical address, or if not resident, its file and offset.

Other options translate the flags of a `vm_area_struct`, or display the full contents of a task's `mm_struct` or of each `vm_area_struct`. It can be called from the `foreach` wrapper command.

vtop — This context-sensitive command translates a user or kernel virtual address to its physical address. Also displayed are:

- the full PTE translation from page directory through to the page table
- the `vm_area_struct` data for user virtual addresses
- the `mem_map` page data associated with the physical page
- the swap location or file location if a user virtual page is not currently mapped

It can be called from the `foreach` wrapper command.

waitq — Lists the tasks linked on a specified kernel wait queue.

Utility Functions

The following commands are a set of useful helper commands serving various purposes, some simple, others quite powerful.

Command	Description
ascii	Translates a numerical value into its ASCII components; with no arguments, displays an ASCII chart.
btop	Translates a byte value (physical address) to its page number.
eval	A simple calculator, evaluates an expression and displays the result in hexadecimal, decimal, octal and binary, and optionally showing the bit numbers set in the result.
list	Dumps the entries of a linked list of structures. It can handle lists of structures that are singly-linked with simple "next" pointers, or those with embedded `list_head` structures. The output may be constrained to simply display the address of each structure in the list, or if directed, also dump each complete structure, or just one member of each structure. The gathered list entries are hashed, so a corrupted list that loops back upon itself will be recognized.
ptob	translates a page frame number to its byte value (physical address).
ptov	Translates a physical address into a kernel virtual address by adding the appropriate `PAGE_OFFSET` value.
search	Searches a range of user or kernel memory space for given value, with an optional "don't care" bit-mask argument.
rd	Displays a specified amount of user virtual, kernel virtual, or physical memory in several formats, such as 8, 16, 32 or 64 bit values,

	hexadecimal or decimal, symbolically, and with ASCII translations. When reading user virtual addresses, the command is context-sensitive.
wr	Modifies the contents of memory on a live system. Write permission on /dev/mem is required; this command should obviously be used with great care. The write operation is constrained to one 8, 16, 32 or 64 bit location.

Session Control Commands

The following commands typcally aid in the efficient running of a crash session.

Command	Description
alias	Creates a single-word alias for a command string. Several aliases are built into crash; user-defined aliases may also be defined in a .crashrc file, or during a crash session by entering it on the command line or reading it from an input file.
exit	Shuts down the crash session (same as q).
extend	Extend the crash command set by dynamically loading a shared object library containing one or more user-written commands.
foreach	Quite often it is helpful, or even necessary, to run the same crash context-sensitive command on a number of tasks by just entering one command. This wrapper command sets off the execution of a given crash command on each of a defined set of tasks, temporarily changing the current context to that of the targeted task before running the command. The set of tasks that are issued the given command can be defined by: ◦ one or more PID numbers ◦ one or more task numbers ◦ one or more command name ◦ all user tasks ◦ all kernel tasks ◦ the active task on each CPU The identifiers above may be mixed if it makes sense, such as using a combination of PIDs, task addresses, and command names. The context-sensitive commands that can be issued to the selected tasks are: ◦ bt ◦ vm ◦ task ◦ files ◦ net ◦ set ◦ sig ◦ vtop A header containing the PID, task address, CPU and command name will be pre-pended before the command output for each selected task.

gdb	This command passes its arguments directly to `gdb` for processing. This is typically not necessary, but where ambiguities between `crash` and `gdb` command names exist, this will force the command to be executed by `gdb`.
repeat	This wrapper command repeats a `crash` command indefinitely, optionally delaying a given number of seconds between each command execution. Obviously this command is only useful when running on a live system.
set	This primary purpose for this command is to set the `crash` context to a new task, or to display the current context. It can also be used to view or change one of a set of internal `crash` variables that modify program behavior, such as the default output radix or scrolling behavior. It can be called from the `foreach` wrapper command for viewing the context data of each task.
q	Shuts down the `crash` session (same as `exit`).

Crash Usage: A Case Study

The steps taken to debug a kernel crash dump are not etched in stone, and the `crash` commands used to debug a kernel issue vary according to the problem exhibited. The section contains of a case study that shows how the capabilities of the `crash` utility were used to to debug a specific kernel problem. However, before doing so, it should be noted that the following commands are typically the most commonly-used:

bt	Display the backtrace of the current context, or as specified with arguments. This command is typically the first command entered after starting a dumpfile session. Since the initial context is the panic context, it will show the function trace leading up to the kernel panic. `bt -a` will show the trace of the *active* task on each CPU, since there may be an interrelationship between the panicking task on one CPU and the running task(s) on the other CPU(s). When `bt` is given as the argument to `foreach`. displays the backtraces of *all* tasks.

struct	Print the contents of a data structure at a specified address. This command is so common that it is typically unnecessary to enter the `struct` command name on the command line; if the first command line argument is not a `crash` or `gdb` command, but it *is* the name of a known data structure, then all the command line arguments are passed to the `struct` command. So for example, the following two commands yield the same result:
	
	`crash> struct vm_area_struct d3cb2600`
	
	`crash> vm_area_struct d3cb2600`

	Set a new task context by PID, task address, or cpu. Since

set	several `crash` commands are context-sensitive, it's helpful to be able to change the context to avoid having to pass the PID or task address to those context-sensitive commands in order to access the data of a task that is *not* the current context.
p	Prints the contents of a kernel variable; since it's a gateway to the `print` command of the mbedded `gdb` module, it can also be used to print complex C language expressions.
rd	Read memory, which may be either kernel virtual, user virtual, or physical, and display it several different formats and sizes.
ps	Lists basic task information for each process; it can also display parent and child hierarchies.
log	Dump the kernel `log_buf`, which often contains clues leading up to a subsequent kernel crash.
foreach	Execute a `crash` command on all tasks, or those specified, in the system; can be used with bt, vm, task, files, net, set, sig and vtop.
files	Dump the open file descriptor data of a task; most usefully, the `file`, `dentry` and `inode` structure addresses for each open file descriptor.
vm	Dump the virtual memory map of a task, including the vital information concerning each `vm_area_struct` making up a task's address space. It can also dump the physical address of each page in the address space, or if not mapped, its location in a file or on the swap device.

A Case Study: `"kernel BUG at pipe.c:120!"`

Upon bringing up a `crash` session, a great deal of information can be gained just by the invocation data. Here is what what displayed in this particular case:

```
...
      KERNEL: vmlinux-2.4.9-e.10.13enterprise-g
    DUMPFILE: vmcore-incomplete
        CPUS: 2
        DATE: Mon Feb 17 08:20:56 2003
      UPTIME: 4 days, 20:04:41
LOAD AVERAGE: 0.95, 1.04, 1.25
       TASKS: 110
    NODENAME: testbox.redhat.com
     RELEASE: 2.4.9-e.10.13enterprise
     VERSION: #1 SMP Mon Feb 3 12:59:26 EST 2003
     MACHINE: i686  (2788 Mhz)
      MEMORY: 6 GB
       PANIC: "kernel BUG at pipe.c:120!"
         PID: 20571
```

```
     COMMAND: "imp"
        TASK: d1566000
         CPU: 1
       STATE: TASK_RUNNING (PANIC)

crash>
```

In this case the `PANIC` string `"kernel BUG at pipe.c:120!"` points to the exact kernel source code line at which the panic occurred.

Then, getting a backtrace of panicking task is typically the first order of the day:

```
crash> bt
PID: 20571  TASK: d1566000  CPU: 1   COMMAND: "imp"
 #0 [d1567e44] die at c010785c
 #1 [d1567e54] do_invalid_op at c0107b2c
 #2 [d1567f0c] error_code (via invalid_op) at c01073de
    EAX: 0000001d  EBX: ed87b2e0  ECX: c02f6064  EDX: 00005fa1  EBP: 00001000
    DS:  0018      ESI: f640e740  ES:  0018      EDI: 00001000
    CS:  0010      EIP: c0150b6d  ERR: ffffffff  EFLAGS: 00010292
 #3 [d1567f48] pipe_read at c0150b6d
 #4 [d1567f6c] sys_read at c01468d4
 #5 [d1567fc0] system_call at c01072dc
    EAX: 00000003  EBX: 0000000a  ECX: 40b4e05c  EDX: 00002000
    DS:  002b      ESI: 00002000  ES:  002b      EDI: 40b4e05c
    SS:  002b      ESP: bffe9e88  EBP: bffe9eb8
    CS:  0023      EIP: 40aaa1d4  ERR: 00000003  EFLAGS: 00000286
```

The backtrace shows that the call to `die()` was generated by an `invalid_op` exception. The exception was caused by the `BUG()` call in the `pipe_read()` function:

```
        if (count && PIPE_WAITING_WRITERS(*inode) &&
                !(filp->f_flags & O_NONBLOCK)) {
                /*
                 * We know that we are going to sleep: signal
                 * writers synchronously that there is more
                 * room.
                 */
                wake_up_interruptible_sync(PIPE_WAIT(*inode));
                if (!PIPE_EMPTY(*inode))
                        BUG();
                goto do_more_read;
        }
```

In the code segment above, the `pipe_read()` code has previously `down`'d the semaphore of the inode associated with the pipe, giving it exclusive access. It had read all data in the pipe, but still needed more to satisfy the `count` requested. Finding that there was a writer with more data -- and who was waiting on the semaphore -- it woke up the writer. However, after doing the wakeup, it did a sanity-check on the pipe contents, and found that it was no longer empty -- which is theoretically *impossible* since it was *still* holding the semaphore. It appeared that the writer process wrote to the pipe while the reader process still had exclusive access -- somehow overriding the semaphore.

Since the semaphore mechanism was seemingly not working, it was first necessary to look at the actual `semaphore` structure associated with the pipe's inode. This first required looking at the first argument to the `pipe_read()` function; the `whatis` command shows that it is a `struct file` pointer:

```
crash> whatis pipe_read
ssize_t pipe_read(struct file *, char *, size_t, loff_t *);
crash>
```

Using the bt -f option, each frame in the backtrace is expanded to show all stack data in the frame. Looking at the expansion of the sys_read() frame, we can see that the last thing pushed on the stack before calling pipe_read() was the file pointer address of edf3f740:

```
...
#3 [d1567f48] pipe_read at c0150b6d
    [RA: c01468d6  SP: d1567f4c  FP: d1567f6c  SIZE: 36]
    d1567f4c: c026701c  00000078  fffffff2  00001000
    d1567f5c: 00000000  edf3f740  ffffffea  00002000
    d1567f6c: c01468d6
 #4 [d1567f6c] sys_read at c01468d4
    [RA: c01072e3  SP: d1567f70  FP: d1567fc0  SIZE: 84]
    d1567f70: edf3f740  40b4f05c  00002000  edf3f760
    d1567f80: c03683d0  fffffffb  00000001  c0120d3b
    d1567f90: 00000046  00000046  0000000b  c0350960
    d1567fa0: 0000000b  f639eb00  c0108e0e  00000020
    d1567fb0: d1566000  00002000  40b4e05c  bffe9eb8
    d1567fc0: c01072e3
...
```

The task at hand is finding the inode containing the suspect semaphore from the file structure address. The file structure's f_dentry member points to its dentry structure, whose d_inode member in turn points to the pipe's inode. The struct command can be used to dump the complete contents of a data structure at a given address; by tagging the .member onto the structure name, we can print just the member desired. By following the structure chain, the inode address can be determined like so:

```
crash> struct file.f_dentry edf3f740
  f_dentry = 0xdb0ec440,
crash> struct dentry.d_inode db0ec440
  d_inode = 0xf640e740,
crash> struct inode.i_sem f640e740

  i_sem = {
    count = {
      counter = 2
    },
    sleepers = 0,
    wait = {
      lock = {
        lock = 1
      },
      task_list = {
        next = 0xf640e7ac,
        prev = 0xf640e7ac
      }
    }
  },
crash>
```

The dump of the semaphore structure above showed the problem: the counter value of 2 is illegal. It should never be greater than 1; in this case a value of 2 allows *two* successful down operations, i.e., giving two tasks access to the pipe at the same time.

(As an aside, determining the inode address above could also be accomplished by using the context-sensitive files command, which dumps the associated file, dentry and inode structure addresses for each open file descriptor of a task. The dumped file descriptor list would contain one with a reference to the file structure at edf3f740, and would also show the associated inode address of f640e740.)

Before getting a dumpfile, this same panic had occurred several times. It was erroneously presumed that the problem was in the pipe-handling code, but it was eventually determined not to be the case. By instrumenting a kernel with debug code, the starting counter value of a pipe was found to be 3. Compounding that problem was the fact that the

inode slab cache is one of a few special cases that presume that the freed inode's contents are left in a legitimate state so that they do not have to be completely reinitialized with each subsequent reallocation. So when the pipe's inode was created, it received an inode with a bogus counter value.

Confirming the existence of bogus inode structures in the slab cache was a multi-stepped procedure. Using the command `kmem` command to access the inode slab cache, we can get the addresses of all free and currently-allocated inodes. Since there are typically several thousand inodes, the output is extremely verbose, but here is the beginning of it:

```
crash> kmem -S inode_cache
CACHE    NAME                 OBJSIZE  ALLOCATED     TOTAL  SLABS  SSIZE
c7666564 inode_cache              448      11563     12339   1371     4k
SLAB       MEMORY    TOTAL  ALLOCATED  FREE
d1d82000  d1d82040      9          9     0
FREE / [ALLOCATED]
  [d1d82040]
  [d1d82200]
  [d1d823c0]
  [d1d82580]
  [d1d82740]
  [d1d82900]
  [d1d82ac0]
  [d1d82c80]
  [d1d82e40]
SLAB       MEMORY    TOTAL  ALLOCATED  FREE
f4e52000  f4e52040      9          7     2
FREE / [ALLOCATED]
   f4e52040  (cpu 1 cache)
   f4e52200  (cpu 1 cache)
  [f4e523c0]
  [f4e52580]
  [f4e52740]
  [f4e52900]
  [f4e52ac0]
  [f4e52c80]
  [f4e52e40]
...
```

In the truncated output above, *all* of the inode address in the slab cache are dumped; the ones currently in use are surrounded by brackets, the free ones are *not*. So, for example, the inodes at addresses `f4e52040` and `f4e52200` are free; the others are not. The full output was piped to a script that pulled out just the free inode addresses (i.e., output lines starting with three spaces), and redirected them into a file. The file was modified to be a `crash` input file by making each extracted inode address to be the arguments of the `struct` command, using its short-cut method that allows the dropping of the `struct` command name; therefore the input file contained hundreds of `crash` commands of the form:

```
inode.i_sem f4e52040
inode.i_sem f4e52200
inode.i_sem f5cdc040
inode.i_sem f5cdc200
inode.i_sem f5cdc3c0
inode.i_sem f5cdc580
...
```

Note that the `struct` command would be used by default above, as documented in its help page; if the first command line argument is not a `crash` or `gdb` command, but it *is* the name of a known data structure, it passes the arguments to the `struct` command.

Using the capability of feeding an input file, in this case consisting of hundreds of short-cut `struct` commands like those above, the output was again quite verbose, consisting of structure member dumps of the form:

```
crash> < input.file
crash> inode.i_sem f4e52040
  i_sem = {
    count = {
      counter = 1
    },
    sleepers = 0,
    wait = {
      lock = {
        lock = 1
      },
      task_list = {
        next = 0xf4e520ac,
        prev = 0xf4e520ac
      }
    }
  },
crash> inode.i_sem f4e52200
  i_sem = {
    count = {
      counter = 1
    },
    sleepers = 0,
    wait = {
      lock = {
        lock = 1
      },
      task_list = {
        next = 0xf4e5226c,
        prev = 0xf4e5226c
      }
    }
  },
...
```

However, it was a simple matter of piping the output to `grep`, and looking for `counter` values not equal to `1`:

```
crash> < input.file | grep counter | grep -v "= 1"
      counter = 3
      counter = 3
      counter = 3
      counter = 3
crash>
```

This turned out to be the smoking gun. Another round of debugging with an instrumented kernel that trapped attempts to free an inode with a semaphore counter of `3` caught the perpetrator in the act.

Command Extensions

Frequently users wish to add an additional option to an existing `crash` command, or add a brand new command, in order to address a kernel issue they are debugging or developing. For those reasons, the `crash` utility was designed with extensibility in mind. There are two ways to add new functionality:

1. adding new code and compiling it into the `crash` executable,
2. creating a shared object library that can be dynamically loaded by using the `extend` command.

This section consists of a quick guide that describes how to get started using both

methods.

Adding new code and compiling it into the `crash` executable

The current set of `crash` commands can be seen by entering the `help` command with no arguments:

```
crash> help

*              files          mod            runq           union
alias          foreach        mount          search         vm
ascii          fuser          net            set            vtop
bt             gdb            p              sig            waitq
btop           help           ps             struct         whatis
dev            irq            pte            swap           wr
dis            kmem           ptob           sym            q
eval           list           ptov           sys
exit           log            rd             task
extend         mach           repeat         timer

crash version: 4.0-8.11   gdb version: 6.1
For help on any command above, enter "help ".
For help on input options, enter "help input".
For help on output options, enter "help output".

crash>
```

For each command in the menu above, there is an entry in a data structure in the file `global_data.c`, which is located in the top-level directory of the `crash` source code tree:

```
/*
 *  To add a new command, declare it in defs.h and enter it in this table.
 */

struct command_table_entry base_command_table[] = {
        {"*",       cmd_pointer, help_pointer, 0},
        {"alias",   cmd_alias,   help_alias,   0},
        {"ascii",   cmd_ascii,   help_ascii,   0},
        {"bt",      cmd_bt,      help_bt,      REFRESH_TASK_TABLE},
        {"btop",    cmd_btop,    help_btop,    0},
        {"dev",     cmd_dev,     help_dev,     0},
        {"dis",     cmd_dis,     help_dis,     0},
        {"eval",    cmd_eval,    help_eval,    0},
        {"exit",    cmd_quit,    help_exit,    0},
        {"extend",  cmd_extend,  help_extend,  0},
        {"files",   cmd_files,   help_files,   REFRESH_TASK_TABLE},
        {"foreach", cmd_foreach, help_foreach, REFRESH_TASK_TABLE},
        {"fuser",   cmd_fuser,   help_fuser,   REFRESH_TASK_TABLE},
        {"gdb",     cmd_gdb,     help_gdb,     REFRESH_TASK_TABLE},
        {"help",    cmd_help,    help_help,    0},
        {"irq",     cmd_irq,     help_irq,     0},
        {"kmem",    cmd_kmem,    help_kmem,    0},
        {"list",    cmd_list,    help__list,   REFRESH_TASK_TABLE},
        {"log",     cmd_log,     help_log,     0},
        {"mach",    cmd_mach,    help_mach,    0},
        {"mod",     cmd_mod,     help_mod,     0},
        {"mount",   cmd_mount,   help_mount,   0},
        {"net",     cmd_net,     help_net,     REFRESH_TASK_TABLE},
        {"p",       cmd_p,       help_p,       0},
        {"ps",      cmd_ps,      help_ps,      REFRESH_TASK_TABLE},
        {"pte",     cmd_pte,     help_pte,     0},
        {"ptob",    cmd_ptob,    help_ptob,    0},
        {"ptov",    cmd_ptov,    help_ptov,    0},
        {"q",       cmd_quit,    help_quit,    0},
        {"rd",      cmd_rd,      help_rd,      0},
        {"repeat",  cmd_repeat,  help_repeat,  0},
        {"runq",    cmd_runq,    help_runq,    REFRESH_TASK_TABLE},
        {"search",  cmd_search,  help_search,  0},
        {"set",     cmd_set,     help_set,     REFRESH_TASK_TABLE},
```

```
        {"sig",     cmd_sig,     help_sig,     REFRESH_TASK_TABLE},
        {"struct",  cmd_struct,  help_struct,  0},
        {"swap",    cmd_swap,    help_swap,    0},
        {"sym",     cmd_sym,     help_sym,     0},
        {"sys",     cmd_sys,     help_sys,     REFRESH_TASK_TABLE},
        {"task",    cmd_task,    help_task,    REFRESH_TASK_TABLE},
        {"test",    cmd_test,    NULL,         HIDDEN_COMMAND},
        {"timer",   cmd_timer,   help_timer,   0},
        {"union",   cmd_union,   help_union,   0},
        {"vm",      cmd_vm,      help_vm,      REFRESH_TASK_TABLE},
        {"vtop",    cmd_vtop,    help_vtop,    REFRESH_TASK_TABLE},
        {"waitq",   cmd_waitq,   help_waitq,   REFRESH_TASK_TABLE},
        {"whatis",  cmd_whatis,  help_whatis,  0},
        {"wr",      cmd_wr,      help_wr,      0},
        {(char *)NULL}
};
```

Each entry consists of the following simple data structure:

```
struct command_table_entry {              /* one for each command in menu */
        char *name;
        cmd_func_t func;
        char **help_data;
        ulong flags;
};
```

The structure members consist of:

`name`	The character string that appears in the help menu output. It must be unique among other `crash` commands, and preferably should not clash with any `gdb` command.
`func`	A pointer to the function that performs the command. To add an option to an existing command, find the file that contains this function, and modify it as desired.
`help_data`	A pointer to an array of character strings that make up the help data for that command. Although optional, it's certainly useful to create help data for any new commands or options.
`flags`	If `REFRESH_TASK_TABLE`, the set of running tasks on a live system will be updated just prior to executing the command. If `HIDDEN_COMMAND`, the command will not be shown in the help menu. The only command with this flag is the `test` command, whose function `cmd_test()` exists solely as an builtin aid for quickly developing new or temporary commands.

For a newly-written command to appear in the help menu, it simply requires a reference to it in the structure above. To test a new command without adding it to the menu, use the hidden "test" command, found in `test.c`:

```
void
cmd_test(void)
{
        int c;

        while ((c = getopt(argcnt, args, "")) != EOF) {
                switch(c)
```

```
                {
                default:
                        argerrs++;
                        break;
                }
        }

        if (argerrs)
                cmd_usage(pc->curcmd, SYNOPSIS);

        while (args[optind]) {
                ;
                optind++;
        }
}
```

The `test` command contains the basic template used by `crash` commands to accept dash-arguments, which are fielded by the `getopt()` routine, while all other command line arguments are fielded in the `while` loop. To add an option to the `test` command (or any other existing command), simply fit it into the appropriate argument-gathering mechanism. Or, for that matter, if no arguments are required, put the functionality at the end of the command's function. Here is a trivial example of a change to the `cmd_test()` function, with the modifications highlighted:

```
void
cmd_test(void)
{
        int c;

        while ((c = getopt(argcnt, args, "xa:")) != EOF) {
                switch(c)
                {
                case 'x':
                        fprintf(fp, "arg: -x\n");
                        break;
                case 'a':
                        fprintf(fp, "arg: -a %s\n", optarg);
                        break;
                default:
                        argerrs++;
                        break;
                }
        }

        if (argerrs)
                cmd_usage(pc->curcmd, SYNOPSIS);

        while (args[optind]) {
                fprintf(fp, "arg: %s\n", args[optind]);
                ;
                optind++;
        }

        fprintf(fp, "do test work here...\n");
}
```

Then re-compile `crash` by entering `make`:

```
# make

TARGET: X86
 CRASH: 4.0-8.11
   GDB: 6.1

cc -c -g -DX86 -D_FILE_OFFSET_BITS=64 build_data.c
cc -c -g -DX86 -D_FILE_OFFSET_BITS=64 test.c
ar -rs crashlib.a main.o tools.o global_data.o memory.o filesys.o help.o task.o
build_data.o kernel.o test.o gdb_interface.o net.o dev.o alpha.o x86.o ppc.o ia6
4.o s390.o s390x.o s390dbf.o ppc64.o x86_64.o extensions.o remote.o va_server.o
```

```
va_server_v1.o symbols.o cmdline.o lkcd_common.o lkcd_v1.o lkcd_v2_v3.o lkcd_v5.
o lkcd_v7.o lkcd_v8.o lkcd_fix_mem.o s390_dump.o netdump.o diskdump.o xendump.o
lkcd_x86_trace.o unwind_v1.o unwind_v2.o unwind_v3.o unwind_x86_32_64.o xen_hype
r.o xen_hyper_command.o xen_hyper_global_data.o xen_hyper_dump_tables.o
gcc -g -O2       \
               -o `cat mergeobj` libgdb.a \
                  ../bfd/libbfd.a ../readline/libreadline.a ../opcodes/libopcod
es.a  ../libiberty/libiberty.a     -lm -lncurses  ../libiberty/libiberty.a  -ldl
 -rdynamic `cat mergelibs`
#
```

Then run the `test` command with the tree possible argument types:

```
crash> test -x -a dasharg this that the other
arg: -x
arg: -a dasharg
arg: this
arg: that
arg: the
arg: other
do test work here...
crash>
```

The easiest way to implement a new command's functionality is to use an existing command as a template. There is a broad range of utility routines that handle argument strings, read memory, access data structures and their members, display data, and so on, that obviate the need to reinvent the wheel to accomplish a task. Look at what other similar commands do, and copy their mechanisms.

Creating a shared object library and loading it with `extend`

While adding a new command and/or command option in the manner above is useful, it would require the same integration mechanism with each subsequent release of the `crash` utility. Since that could become tedious, another extension mechanism exists in which share objects containing one or more `crash` commands can be written, and then dynamically attached to a running `crash` session, using the `extend` command. Once loaded, the command(s) in the shared object library automatically appear in the help menu, as if they were compiled into the `crash` executable. As an quick aid in creating a shared object, the help page for the `extend` contains an example C program tagged onto the end, which adds a new `echo` command (which simply echoes back all arguments). The C program piece can be cut and pasted into a file, say `echo.c` for example, and then compiled like so:

```
# gcc -nostartfiles -shared -rdynamic -o echo.so echo.c -fPIC -D<machine-type> $(TARGET_CFLAGS)
```

where <`machine-type`> is the appropriate architecture `X86, X86_64, IA64, PPC64, S390` or `S390X`), and where `$(TARGET_CFLAGS)` is `-D_FILE_OFFSET_BITS=64` on 32-bit architectures, and `-m64` on ppc64. So for an x86 build, the compile line would be:

```
# gcc -nostartfiles -shared -rdynamic -o echo.so echo.c -fPIC -DX86 -D_FILE_OFFSET_BITS=64
```

The resultant `echo.so` file may be dynamically linked into `crash` during runtime using the `extend` command:

```
crash> extend echo.so
./echo.so: shared object loaded
crash>
```

Or, to automatically load the shared library during `crash` session initialization, put the following string into a `.crashrc` file located in the current directory, or in the user's `$HOME`

directory:

```
extend echo.so
```

Here is the help menu once the library is loaded; note the integration of the new `echo` command:

```
crash> help

*              extend         mach           repeat         timer
alias          files          mod            runq           union
ascii          foreach        mount          search         vm
bt             fuser          net            set            vtop
btop           gdb            p              sig            waitq
dev            help           ps             struct         whatis
dis            irq            pte            swap           wr
echo           kmem           ptob           sym            q
eval           list           ptov           sys
exit           log            rd             task

crash version: 4.0-8.11   gdb version: 6.1
For help on any command above, enter "help ".
For help on input options, enter "help input".
For help on output options, enter "help output".
```

With this extension mechanism, the most that would be required to use the shared library with subsequent versions of `crash` would be a simple re-compile of the `echo.c` file.

---

contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section

of the Document). You may use the same title as a previous version
if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities
responsible for authorship of the modifications in the Modified
Version, together with at least five of the principal authors of the
Document (all of its principal authors, if it has fewer than five),
unless they release you from this requirement.
C. State on the Title page the name of the publisher of the
Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications
adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice
giving the public permission to use the Modified Version under the
terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections
and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add
to it an item stating at least the title, year, new authors, and
publisher of the Modified Version as given on the Title Page. If
there is no section Entitled "History" in the Document, create one
stating the title, year, authors, and publisher of the Document as
given on its Title Page, then add an item describing the Modified
Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for
public access to a Transparent copy of the Document, and likewise
the network locations given in the Document for previous versions
it was based on. These may be placed in the "History" section.
You may omit a network location for a work that was published at
least four years before the Document itself, or if the original
publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications",
Preserve the Title of the section, and preserve in the section all
the substance and tone of each of the contributor acknowledgements
and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document,
unaltered in their text and in their titles. Section numbers
or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section
may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements"
or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or
appendices that qualify as Secondary Sections and contain no material
copied from the Document, you may at your option designate some or all
of these sections as invariant. To do this, add their titles to the
list of Invariant Sections in the Modified Version's license notice.
These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains
nothing but endorsements of your Modified Version by various
parties--for example, statements of peer review or that the text has
been approved by an organization as the authoritative definition of a
standard.

You may add a passage of up to five words as a Front-Cover Text, and a
passage of up to 25 words as a Back-Cover Text, to the end of the list
of Cover Texts in the Modified Version. Only one passage of
Front-Cover Text and one of Back-Cover Text may be added by (or
through arrangements made by) any one entity. If the Document already
includes a cover text for the same cover, previously added by you or
by arrangement made by the same entity you are acting on behalf of,
you may not add another; but you may replace the old one, on explicit
permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License
give permission to use their names for publicity for or to assert or
imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual

title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.