JMake: Dependable Compilation for Kernel Janitors

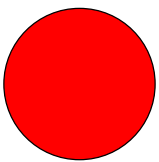
Julia Lawall, Gilles Muller
Inria/LIP6-Whisper

June 28, 2017

Software grows over time

Python v0.9.8:
61K LOC
1993

Wine v0.0.2:
2K LOC
1993

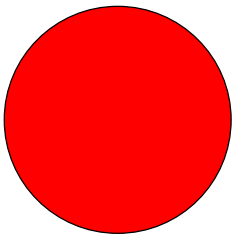


Linux v1.0:
122K LOC
1994

Software grows over time

Python v2.7:
850K LOC
2010

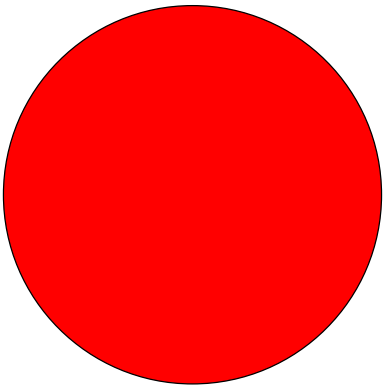
Wine v1.0:
1.5M LOC
2008



Linux v3.0:
10M LOC
2011

Software grows over time

Python v3.6.1:
982K LOC
2017

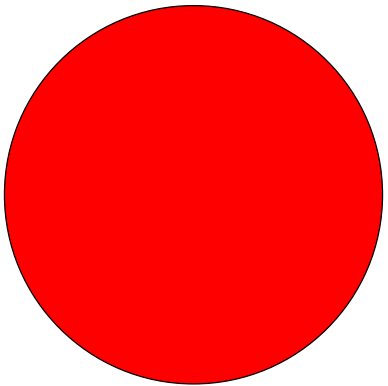


Wine v2.11:
2.8M LOC
2017

Linux v4.11:
15M LOC
2017

Software grows over time

Python v3.6.1:
982K LOC
2017



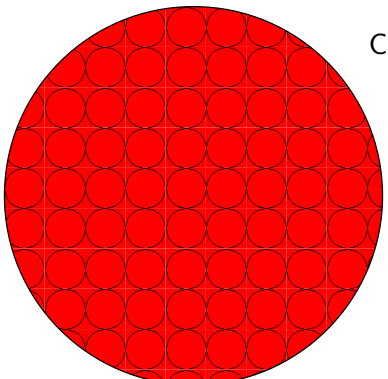
Wine v2.11:
2.8M LOC
2017

Linux v4.11:
15M LOC
2017

Need to support different configurations

CONFIG_ARM

CONFIG_PM_SLEEP

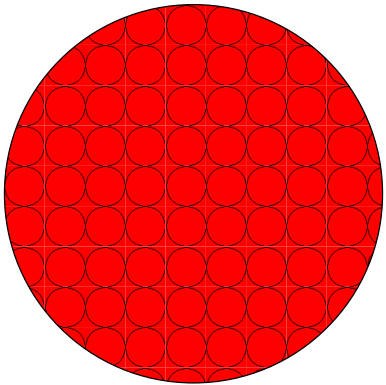


CONFIG_MIPS_GIC

Need for support from different kinds of developers

Maintainers

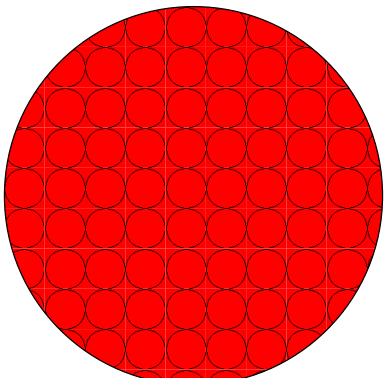
Contributors



Janitors

Need for support from different kinds of developers

Maintainers



Contributors

Janitors

Janitor definition

A SoftwareJanitor is a GruntProgrammer who comes in to clean up the other developers' messes or do otherwise necessary yet unglamorous tasks.

http://wiki.c2.com/?SoftwareJanitor

Janitor definition

A SoftwareJanitor is a GruntProgrammer who comes in to clean up the other developers' messes or do otherwise necessary yet unglamorous tasks.

http://wiki.c2.com/?SoftwareJanitor

- Janitors know coding style conventions and API changes.
- Janitors may not know individual subsystems deeply.
- Testing often limited to successful compilation.

Silent compiler failure scenario

- Janitor modifies some code.
- Compilation succeeds.
- But errors may remain, if the configuration chosen does not subject the changed lines to compilation.

This work

Goal: Improve the reliability of the work of janitors.

This work

Goal: Improve the reliability of the work of janitors.

Our approach: JMake

- Automate the choice of architecture and configuration
- Automate the detection of lines subjected to the compiler

This work

Goal: Improve the reliability of the work of janitors.

Our approach: JMake

- Automate the choice of architecture and configuration
- Automate the detection of lines subjected to the compiler

Want to provide immediate feedback, via a tool that janitors can run themselves.

Example (commit 538ea4a)

```
diff --git a/kernel/memremap.c b/kernel/memremap.c
--- a/kernel/memremap.c
+++ b/kernel/memremap.c
@@ -114,7 +114,8 @@
 {
        void **ptr, *addr;

-       ptr = devres_alloc(devm_memremap_release, sizeof(*ptr), GFP_KERNEL);
+       ptr = devres_alloc_node(devm_memremap_release, sizeof(*ptr), GFP_KERNEL,
+                               dev_to_node(dev));
        if (!ptr)
                return ERR_PTR(-ENOMEM);

@@ -165,8 +166,8 @@
        if (is_ram == REGION_INTERSECTS)
                return __va(res->start);

-       page_map = devres_alloc(devm_memremap_pages_release,
-                               sizeof(*page_map), GFP_KERNEL);
+       page_map = devres_alloc_node(devm_memremap_pages_release,
+                               sizeof(*page_map), GFP_KERNEL, dev_to_node(dev));
        if (!page_map)
                return ERR_PTR(-ENOMEM);
```

Example (commit 538ea4a)

```
diff --git a/kernel/memremap.c b/kernel/memremap.c
--- a/kernel/memremap.c
+++ b/kernel/memremap.c
@@ -114,7 +114,8 @@
 {
        void **ptr, *addr;

-       ptr = devres_alloc(devm_memremap_release, sizeof(*ptr), GFP_KERNEL);
+       ptr = devres_alloc_node(devm_memremap_release, sizeof(*ptr), GFP_KERNEL,
+                               dev_to_node(dev));
        if (!ptr)
                return ERR_PTR(-ENOMEM);

@@ -165,8 +166,8 @@
        if (is_ram == REGION_INTERSECTS)
                return __va(res->start);

-       page_map = devres_alloc(devm_memremap_pages_release,
-                               sizeof(*page_map), GFP_KERNEL);
+       page_map = devres_alloc_node(devm_memremap_pages_release,
+                               sizeof(*page_map), GFP_KERNEL, dev_to_node(dev));
        if (!page_map)
                return ERR_PTR(-ENOMEM);
```

- First change compiled for x86/allyesconfig.
- Second under `#ifdef CONFIG_ZONE_DEVICE` (29 lines up)

Example (commit 538ea4a)

```
diff --git a/kernel/memremap.c b/kernel/memremap.c
--- a/kernel/memremap.c
+++ b/kernel/memremap.c
@@ -114,7 +114,8 @@
 {
        void **ptr, *addr;

-       ptr = devres_alloc(devm_memremap_release, sizeof(*ptr), GFP_KERNEL);
+       ptr = devres_alloc_node(devm_memremap_release, sizeof(*ptr), GFP_KERNEL,
+                               dev_to_node(dev));
        if (!ptr)
                return ERR_PTR(-ENOMEM);

@@ -165,8 +166,8 @@
        if (is_ram == REGION_INTERSECTS)
                return __va(res->start);

-       page_map = devres_alloc(devm_memremap_pages_release,
-                               sizeof(*page_map), GFP_KERNEL);
+       page_map = devres_alloc_node(devm_memremap_pages_release,
+                               sizeof(*page_map), GFP_KERNEL, dev_to_node(dev));
        if (!page_map)
                return ERR_PTR(-ENOMEM);
```

- First change compiled for x86/allyesconfig.
- Second under `#ifdef CONFIG_ZONE_DEVICE` (29 lines up)
  - JMake reports that the second is not compiled.

Example (commit 7d32cde)

```
diff --git a/drivers/usb/musb/musb_core.c b/drivers/usb/musb/musb_core.c
--- a/drivers/usb/musb/musb_core.c
+++ b/drivers/usb/musb/musb_core.c
@@ -2094,6 +2094,7 @@
 #ifndef CONFIG_MUSB_PIO_ONLY
        if (!musb->ops->dma_init || !musb->ops->dma_exit) {
                dev_err(dev, "DMA controller not set\n");
+               status = -ENODEV;
                goto fail2;
        }
        musb_dma_controller_create = musb->ops->dma_init;
```

Example (commit 7d32cde)

```
diff --git a/drivers/usb/musb/musb_core.c b/drivers/usb/musb/musb_core.c
--- a/drivers/usb/musb/musb_core.c
+++ b/drivers/usb/musb/musb_core.c
@@ -2094,6 +2094,7 @@
 #ifndef CONFIG_MUSB_PIO_ONLY
        if (!musb->ops->dma_init || !musb->ops->dma_exit) {
                dev_err(dev, "DMA controller not set\n");
+               status = -ENODEV;
                goto fail2;
        }
        musb_dma_controller_create = musb->ops->dma_init;
```

- Compilation succeeds for x86/allyesconfig, but JMake reports that the changed line is overlooked.

Example (commit 7d32cde)

```
diff --git a/drivers/usb/musb/musb_core.c b/drivers/usb/musb/musb_core.c
--- a/drivers/usb/musb/musb_core.c
+++ b/drivers/usb/musb/musb_core.c
@@ -2094,6 +2094,7 @@
 #ifndef CONFIG_MUSB_PIO_ONLY
        if (!musb->ops->dma_init || !musb->ops->dma_exit) {
                dev_err(dev, "DMA controller not set\n");
+               status = -ENODEV;
                goto fail2;
        }
        musb_dma_controller_create = musb->ops->dma_init;
```

- Compilation succeeds for x86/allyesconfig, but JMake reports that the changed line is overlooked.
- JMake finds that the changed line is compiled for ARM.

Step 1: Choice of architecture and configuration

Key observation: Compilation is architecture (`arch`) specific.

Available resources:

- Linux kernel cross-compilation infrastructure
- Provided sample configurations

Step 1: Choice of architecture and configuration

Key observation: Compilation is architecture (`arch`) specific.

Available resources:

- Linux kernel cross-compilation infrastructure
- Provided sample configurations

Issue: Compilation is expensive

- 24 architectures supported.
- $\sim$500 sample configurations provided.
- Infeasible to consider them all.

Choice of architecture and configuration for .c files

Search heuristics:

1. Architecture-specific allyesconfig for `arch` files,
   Default x86/allyesconfig for others.
2. CONFIG variable from Makefile line for the .c file
   - allyesconfig for each `arch` that references CONFIG
   - one specific config file, if any in that `arch`, that references CONFIG
3. All CONFIG variables in the Makefile referencing the .c file
   - Same as in the previous case.

Choice of architecture and configuration for `.h` files

Extra challenges:

- `.h` files cannot be compiled directly.
- Need to find a `.c` file that is affected by the changes in the `.h` file.
- Multiple header files may have the same name - inclusion is configuration dependent.
  - Select all `.c` files including files with the header name.
- Header files often define macros, which are only subject to compilation if used.
  - Prioritize `.c` files that refer to changed macros.

Step 2: Detecting which lines are subjected to the compiler

Issue: Due to config options, compilation of a changed file can succeed without checking the changes.

Step 2: Detecting which lines are subjected to the compiler

Issue: Due to config options, compilation of a changed file can succeed without checking the changes.

Options:

- Check line numbers in compiled code (e.g., `.lst` file).

Step 2: Detecting which lines are subjected to the compiler

Issue: Due to config options, compilation of a changed file can succeed without checking the changes.

Options:

- Check line numbers in compiled code (e.g., `.lst` file).
  - Macro bodies move to usage points, lose line numbers.

Step 2: Detecting which lines are subjected to the compiler

Issue: Due to config options, compilation of a changed file can succeed without checking the changes.

Options:

- Check line numbers in compiled code (e.g., `.lst` file).
  - Macro bodies move to usage points, lose line numbers.
- Mutate changed source code, look for line numbers in error messages.

Step 2: Detecting which lines are subjected to the compiler

Issue: Due to config options, compilation of a changed file can succeed without checking the changes.

Options:

- Check line numbers in compiled code (e.g., `.lst` file).
  - Macro bodies move to usage points, lose line numbers.
- Mutate changed source code, look for line numbers in error messages.
  - No control of the compiler's error reporting strategy.

Step 2: Detecting which lines are subjected to the compiler

Issue: Due to config options, compilation of a changed file can succeed without checking the changes.

Options:

- Check line numbers in compiled code (e.g., `.lst` file).
  - Macro bodies move to usage points, lose line numbers.
- Mutate changed source code, look for line numbers in error messages.
  - No control of the compiler's error reporting strategy.

Our solution:

- Mutate changed source code, look for mutations in preprocessed code (`.i` files).

Step 2: Detecting which lines are subjected to the compiler

Issue: Due to config options, compilation of a changed file can succeed without checking the changes.

Options:

- Check line numbers in compiled code (e.g., `.lst` file).
  - Macro bodies move to usage points, lose line numbers.
- Mutate changed source code, look for line numbers in error messages.
  - No control of the compiler's error reporting strategy.

Our solution:

- Mutate changed source code, look for mutations in preprocessed code (`.i` files).
  - Final validation: produce unmutated `.o` file.

Example

Linux kernel commit 95ea3e760ef8:

```
@@ -48,0 +49,4 @@
+#define DAS16CS_AI_MUX_HI_CHAN(x)     (((x) & 0xf) << 4)
+#define DAS16CS_AI_MUX_LO_CHAN(x)     (((x) & 0xf) << 0)
+#define DAS16CS_AI_MUX_SINGLE_CHAN(x)(DAS16CS_AI_MUX_HI_CHAN(x)|\
+                                       DAS16CS_AI_MUX_LO_CHAN(x))

@@ -114 +118,2 @@
-       outw(chan, dev->iobase + DAS16CS_AI_MUX_REG);
+       outw(DAS16CS_AI_MUX_SINGLE_CHAN(chan),
+            dev->iobase + DAS16CS_AI_MUX_REG);
```

Example

Mutated code:

```
#define DAS16CS_AI_MUX_HI_CHAN(x)     (((x) & 0xf) << 4)¤"define:xcb_das16_cs.c:49"¤
#define DAS16CS_AI_MUX_LO_CHAN(x)     (((x) & 0xf) << 0)¤"define:cb_das16_cs.c:50"¤
#define DAS16CS_AI_MUX_SINGLE_CHAN(x) (DAS16CS_AI_MUX_HI_CHAN(x) | ¤"define:cb_das16_cs.c:51"¤ \
                                       DAS16CS_AI_MUX_LO_CHAN(x))
...
¤"noncomment:cb_das16_cs.c:118"¤
outw(DAS16CS_AI_MUX_SINGLE_CHAN(chan),
     dev->iobase + DAS16CS_AI_MUX_REG);
```

Example

Mutated code:

```
#define DAS16CS_AI_MUX_HI_CHAN(x)     (((x) & 0xf) << 4)¤"define:xcb_das16_cs.c:49"¤
#define DAS16CS_AI_MUX_LO_CHAN(x)     (((x) & 0xf) << 0)¤"define:cb_das16_cs.c:50"¤
#define DAS16CS_AI_MUX_SINGLE_CHAN(x) (DAS16CS_AI_MUX_HI_CHAN(x) | ¤"define:cb_das16_cs.c:51"¤ \
                                       DAS16CS_AI_MUX_LO_CHAN(x))
...
¤"noncomment:cb_das16_cs.c:118"¤
outw(DAS16CS_AI_MUX_SINGLE_CHAN(chan),
     dev->iobase + DAS16CS_AI_MUX_REG);
```

Generated .i code:

```
¤"noncomment:cb_das16_cs.c:118"¤
outw(((((chan) & 0xf) << 4)¤"define:xcb_das16_cs.c:49"¤ | ¤"define:cb_das16_cs.c:51"¤
      (((chan) & 0xf) << 0)¤"define:cb_das16_cs.c:50"¤),
     dev->iobase + DAS16CS_AI_MUX_REG);
```

Example

Mutated code:

```
#define DAS16CS_AI_MUX_HI_CHAN(x)     (((x) & 0xf) << 4)¤"define:xcb_das16_cs.c:49"¤
#define DAS16CS_AI_MUX_LO_CHAN(x)     (((x) & 0xf) << 0)¤"define:cb_das16_cs.c:50"¤
#define DAS16CS_AI_MUX_SINGLE_CHAN(x) (DAS16CS_AI_MUX_HI_CHAN(x) | ¤"define:cb_das16_cs.c:51"¤ \
                                       DAS16CS_AI_MUX_LO_CHAN(x))
...
¤"noncomment:cb_das16_cs.c:118"¤
outw(DAS16CS_AI_MUX_SINGLE_CHAN(chan),
     dev->iobase + DAS16CS_AI_MUX_REG);
```

Generated .i code:

```
¤"noncomment:cb_das16_cs.c:118"¤
outw(((((chan) & 0xf) << 4)¤"define:xcb_das16_cs.c:49"¤ | ¤"define:cb_das16_cs.c:51"¤
      (((chan) & 0xf) << 0)¤"define:cb_das16_cs.c:50"¤),
     dev->iobase + DAS16CS_AI_MUX_REG);
```

All changes compiled in this case.

Evaluation

Data: All commits in Linux kernel v4.3..v4.4 (2.5 months, Nov 2015 - Jan 2016)

- 11K commits considered

Test machine: 48-core AMD Opteron 6172, 2.1 GHz CPUs, 12 512KB L2 caches, and 251G RAM.

- Each commit processed on a single core.

Benefits of alternate compilations

- Most files are affected by changes that benefit from compilation for x86_64: 17091 (96%)
- 365 non-arch .c files do not benefit from compilation for x86_64, but do benefit from compilation for some other architecture.
  - Typically ARM.
- 75 non-arch .h files do not benefit from compilation for x86_64, but do benefit from compilation for some other architecture.

Silent compiler failures

415 (3%) of `.c` file instances compile successfully with `make allyesconfig`, but not all modified lines are subjected to the compiler.

- For 54 of these file instances, JMake ultimately succeeds, by considering other architectures.
- For 361 file instances, JMake reports failure.
- JMake is beneficial in both cases.

Silent compiler failures

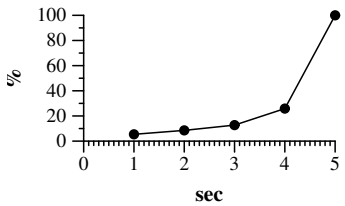
415 (3%) of `.c` file instances compile successfully with `make allyesconfig`, but not all modified lines are subjected to the compiler.

- For 54 of these file instances, JMake ultimately succeeds, by considering other architectures.
- For 361 file instances, JMake reports failure.
- JMake is beneficial in both cases.

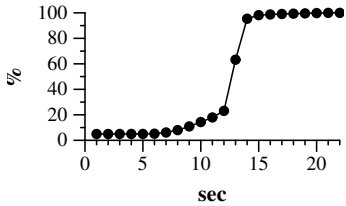
Some issues:

- Never set configuration variable.
- Changes in both #ifdef and #else.
- Changes under #ifdef MODULE
- Changes in unused macros.

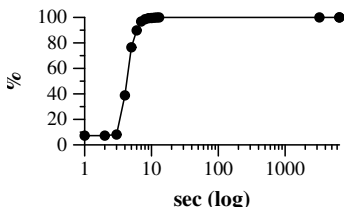
Execution time



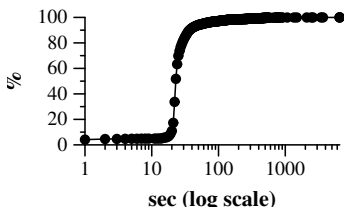
a) make allyesconfig



b) make .i



c) make .o



d) overall

11K commits
2.1 GHz CPU

What about janitors?

A SoftwareJanitor is a GruntProgrammer who comes in to clean up the other developers' messes or do otherwise necessary yet unglamorous tasks.

What about janitors?

> *A SoftwareJanitor is a GruntProgrammer who comes in to clean up the other developers' messes or do otherwise necessary yet unglamorous tasks.*

Need a more quantitative definition...

Proposed janitor characterization

Some thresholds (v3.0..v4.4):

# patches	≥ 10
# subsystems	≥ 20
# lists	≥ 3
# maintainer patches	$\leq 5\%$

File coefficient of variation (cv):

$$\frac{\text{Standard deviation in commits per modified file}}{\text{Mean commits per modified file.}}$$

Identified janitors (top 10 by lowest cv)

	patches	subsystems	lists	maintainer	file cv
Javier Martinez Canillas	118	61	30	0%	0.25
Luis de Bethencourt	104	56	31	0%	0.41
Dan Carpenter (T)	1554	400	146	0%	0.43
Julia Lawall (T)	653	255	93	0%	0.67
Shraddha Barke (I)	160	21	14	0%	0.72
Joe Perches (T)	1078	530	158	2%	0.81
Axel Lin	1044	142	49	0%	0.92
Daniel Borkmann	121	25	15	0%	1.29
Fabio Estevam	790	95	42	0%	1.29
Jarkko Nikula	173	30	14	0%	1.35

$\sim$600 commits in v4.3..v4.4

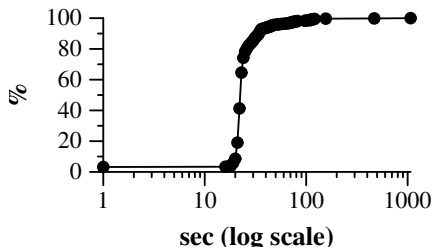
Results on janitor patches

21 silent compilation failures on .c files, 3 on .h files

Results on janitor patches

21 silent compilation failures on .c files, 3 on .h files

Running time on janitor commits (CDF):



Conclusion

- JMake addresses the problem of reliably compile checking Linux kernel code changes.
  - Automatic choice of architecture/configuration.
  - Feedback on the compilation of changed lines, in the presence of conditional compilation.
- Forces compilation of all changed lines on 85% of all commits and 88% of janitor commits.
  - Over 80% of commits treated in 30 seconds or less.
- Potentially applicable to other software for which configurations are available.
- Currently being used in our kernel constification project (CII).

Conclusion

- JMake addresses the problem of reliably compile checking Linux kernel code changes.
  - Automatic choice of architecture/configuration (Linux specific heuristics).
  - Feedback on the compilation of changed lines, in the presence of conditional compilation.
- Forces compilation of all changed lines on 85% of all commits and 88% of janitor commits.
  - Over 80% of commits treated in 30 seconds or less.
- Potentially applicable to other software for which configurations are available.
- Currently being used in our kernel constification project (CII).

Conclusion

- JMake addresses the problem of reliably compile checking Linux kernel code changes.
  - Automatic choice of architecture/configuration (Linux specific heuristics).
  - Feedback on the compilation of changed lines, in the presence of conditional compilation (Linux independent methodology).
- Forces compilation of all changed lines on 85% of all commits and 88% of janitor commits.
  - Over 80% of commits treated in 30 seconds or less.
- Potentially applicable to other software for which configurations are available.
- Currently being used in our kernel constification project (CII).

Conclusion

- JMake addresses the problem of reliably compile checking Linux kernel code changes.
  - Automatic choice of architecture/configuration (Linux specific heuristics).
  - Feedback on the compilation of changed lines, in the presence of conditional compilation (Linux independent methodology).
- Forces compilation of all changed lines on 85% of all commits and 88% of janitor commits.
  - Over 80% of commits treated in 30 seconds or less.
- Potentially applicable to other software for which configurations are available.
- Currently being used in our kernel constification project (CII).

http://jmake-release.gforge.inria.fr