Staging of Artifacts in a Build System

Sascha Roloff
sascha.roloff@huawei.com

Intelligent Cloud Technologies Lab, Huawei Munich Research Center

FOSDEM 2023

Make example

A simple hello world program to generate some sample output, built with BSD make

```
$ cat Makefile
main: main.out.txt

hello: hello.o greet.a
    $(CXX) $(.ALLSRC) -o $(.TARGET)

hello.o: hello.cpp greet.hpp
    $(CXX) -c $(.ALLSRC:[1]) -o $(.TARGET)

greet.a: greet.o
    $(AR) cqs $(.TARGET) $(.ALLSRC:[1])

greet.o: greet.cpp greet.hpp
    $(CXX) -c $(.ALLSRC:[1]) -o $(.TARGET) -DWHOM=\"World\"

main.out.txt: hello
    ./hello > $(.TARGET)
$
```

Make example

A simple hello world program to generate some sample output, built with BSD make

```
$ cat Makefile
main: main.out.txt

hello: hello.o greet.a
    $(CXX) $(.ALLSRC) -o $(.TARGET)

hello.o: hello.cpp greet.hpp
    $(CXX) -c $(.ALLSRC:[1]) -o $(.TARGET)

greet.a: greet.o
    $(AR) cqs $(.TARGET) $(.ALLSRC:[1])

greet.o: greet.cpp greet.hpp
    $(CXX) -c $(.ALLSRC:[1]) -o $(.TARGET) -DWHOM=\"World\"

main.out.txt: hello
    ./hello > $(.TARGET)
$
```

```
$ bmake
g++ -c hello.cpp -o hello.o
g++ -c greet.cpp -o greet.o -DWHOM=\"World\"
ar cqs greet.a greet.o
g++ hello.o greet.a -o hello
./hello > main.out.txt
$

$ cat main.out.txt
Hello World
$
```

Make example

Add some postprocessing to the sample output

```
$ cat Makefile
main: main.out.txt

hello: hello.o greet.a
    $(CXX) $(.ALLSRC) -o $(.TARGET)

hello.o: hello.cpp greet.hpp
    $(CXX) -c $(.ALLSRC:[1]) -o $(.TARGET)

greet.a: greet.o
    $(AR) cqs $(.TARGET) $(.ALLSRC:[1])

greet.o: greet.cpp greet.hpp
    $(CXX) -c $(.ALLSRC:[1]) -o $(.TARGET) -DWHOM=\"World\"

use.txt: hello
    ./hello > $(.TARGET)

postprocessed.txt: use.txt
    tr 'a-z' 'A-Z' < use.txt > postprocessed.txt

main.out.txt: postprocessed.txt
    cat postprocessed.txt > $(.TARGET)
$
```

Make example

Add some postprocessing to the sample output

```
$ cat Makefile
main: main.out.txt

hello: hello.o greet.a
    $(CXX) $(.ALLSRC) -o $(.TARGET)

hello.o: hello.cpp greet.hpp
    $(CXX) -c $(.ALLSRC:[1]) -o $(.TARGET)

greet.a: greet.o
    $(AR) cqs $(.TARGET) $(.ALLSRC:[1])

greet.o: greet.cpp greet.hpp
    $(CXX) -c $(.ALLSRC:[1]) -o $(.TARGET) -DWHOM=\"World\"

use.txt: hello
    ./hello > $(.TARGET)

postprocessed.txt: use.txt
    tr 'a-z' 'A-Z' < use.txt > postprocessed.txt

main.out.txt: postprocessed.txt
    cat postprocessed.txt > $(.TARGET)
$
```

```
$ bmake
g++ -c hello.cpp -o hello.o
g++ -c greet.cpp -o greet.o -DWHOM=\"World\"
ar cqs greet.a greet.o
g++ hello.o greet.a -o hello
./hello > use.txt
tr 'a-z' 'A-Z' < use.txt > postprocessed.txt
cat postprocessed.txt > main.out.txt
$

$ cat main.out.txt
HELLO WORLD
$
```

Make example

Introduce localization as program variants and unite sample output

```
$ cat Makefile
main: main.out.txt

hello.o: hello.cpp greet.hpp
    $(CXX) -c $(.ALLSRC:[1]) -o $(.TARGET)

.for name in Munich Brussels
hello.$(name): hello.o greet.$(name).a
    $(CXX) $(.ALLSRC) -o $(.TARGET)

greet.$(name).a: greet.$(name).o
    $(AR) cqs $(.TARGET) $(.ALLSRC:[1])

greet.$(name).o: greet.cpp greet.hpp
    $(CXX) -c $(.ALLSRC:[1]) -o $(.TARGET) -DWHOM=\"$(name)\"

use.$(name).txt: hello.$(name)
    ./hello.$(name) > $(.TARGET)

postprocessed.$(name).txt: use.$(name).txt
    tr 'a-z' 'A-Z' < use.$(name).txt > postprocessed.$(name).txt
.endfor

main.out.txt: postprocessed.Munich.txt postprocessed.Brussels.txt
    cat $(.ALLSRC) > $(.TARGET)
$
```

Make example

Introduce localization as program variants and unite sample output

```
$ cat Makefile
main: main.out.txt

hello.o: hello.cpp greet.hpp
    $(CXX) -c $(.ALLSRC:[1]) -o $(.TARGET)

.for name in Munich Brussels
hello.$(name): hello.o greet.$(name).a
    $(CXX) $(.ALLSRC) -o $(.TARGET)

greet.$(name).a: greet.$(name).o
    $(AR) cqs $(.TARGET) $(.ALLSRC:[1])

greet.$(name).o: greet.cpp greet.hpp
    $(CXX) -c $(.ALLSRC:[1]) -o $(.TARGET) -DWHOM=\"$(name)\"

use.$(name).txt: hello.$(name)
    ./hello.$(name) > $(.TARGET)

postprocessed.$(name).txt: use.$(name).txt
    tr 'a-z' 'A-Z' < use.$(name).txt > postprocessed.$(name).txt
.endfor

main.out.txt: postprocessed.Munich.txt postprocessed.Brussels.txt
    cat $(.ALLSRC) > $(.TARGET)
$
```

```
$ bmake
g++ -c hello.cpp -o hello.o
g++ -c greet.cpp -o greet.Munich.o -DWHOM=\"Munich\"
ar cqs greet.Munich.a greet.Munich.o
g++ hello.o greet.Munich.a -o hello.Munich
./hello.Munich > use.Munich.txt
tr 'a-z' 'A-Z' < use.Munich.txt > postprocessed.Munich.txt
g++ -c greet.cpp -o greet.Brussels.o -DWHOM=\"Brussels\"
ar cqs greet.Brussels.a greet.Brussels.o
g++ hello.o greet.Brussels.a -o hello.Brussels
./hello.Brussels > use.Brussels.txt
tr 'a-z' 'A-Z' < use.Brussels.txt > postprocessed.Brussels.txt
cat postprocessed.Munich.txt postprocessed.Brussels.txt > main.out.txt
$

$ cat main.out.txt
HELLO MUNICH
HELLO BRUSSELS
$
```

Bazel example

Example application, built with bazel

```
$ cat BUILD
NAMES = ["Munich", "Brussels"]

[cc_binary(
    name = "hello.%s" % (name,),
    srcs = ["hello.cpp"],
    deps = [":greet.%s" % (name,),],
) for name in NAMES]

[cc_library(
    name    = "greet.%s" % (name,),
    hdrs    = ["greet.hpp"],
    srcs    = ["greet.cpp"],
    defines = ["'WHOM=\"%s\"'" % (name,)],
) for name in NAMES]

[genrule(
    name  = "use.%s" % (name,),
    outs  = ["use.%s.txt" % (name,)],
    cmd   = "$(location hello.%s) > $@" % (name,),
    tools = ["hello.%s" % (name,)],
) for name in NAMES]

[genrule(
    name = "postprocessed.%s" % (name,),
    outs = ["postprocessed.%s.txt" % (name,)],
    cmd  = "tr 'a-z' 'A-Z' < $(location use.%s) > $@" % (name,),
    srcs = ["use.%s" % (name,)],
) for name in NAMES]
```

```
genrule(
    name = "main",
    outs = ["main.out.txt"],
    cmd  = "cat $(SRCS) > $@",
    srcs = ["postprocessed.Munich", "postprocessed.Brussels"],
)
$
```

Observation

- Many modern build systems nowadays still follow a design decision implemented by make in the mid 70s

make design decision

Each artifact needs to have a fixed location in the file system

  - Allows to compare timestamps as computationally cheap solution to the problem of *How to determine which parts of a program needs to be recompiled?*
- Once required, today there is no necessity anymore for this restriction
  - Build systems anyway isolate their actions to avoid getting unwanted inputs into their builds
  - Remote execution is also already common practice to take advantage of action distribution and shared caches

Staging

- There is no technical reason for a modern build system to enforce an association of artifacts with the file system
- We propose: Build systems should get over this outdated common practice and apply *staging* instead

What is staging?

Actions can freely and independently select the input and output location of artifacts within their working directory

- Staging stricly separates physical from logical paths
  - Each target has its own view of the world and can place generated artifacts at any logical path they like
  - Consuming targets may place these artifacts at a different logical location
  - All what matters is how the target is defined and not where

Just example

Example application, built with just (build description)

```
$ cat TARGETS
{ "hello":
  { "type": ["@", "rules", "CC", "binary"]
  , "name": ["hello"]
  , "srcs": ["hello.cpp"]
  , "private-deps": ["greet"]
  }
, "greet":
  { "type": ["@", "rules", "CC", "library"]
  , "arguments_config": ["whom"]
  , "name": ["greet"]
  , "hdrs": ["greet.hpp"]
  , "srcs": ["greet.cpp"]
  , "private-defines":
    [ { "type": "join"
      , "$1":
        ["WHOM=\"", {"type": "var", "name": "whom", "default": "World"}, "\""]
      }
    ]
  }
, "use":
  { "type": "generic"
  , "outs": ["use.txt"]
  , "cmds": ["./hello > use.txt"]
  , "deps": ["hello"]
  }
, "postprocessed":
  { "type": "generic"
  , "outs": ["postprocessed.txt"]
  , "cmds": ["tr 'a-z' 'A-Z' < use.txt > postprocessed.txt"]
  , "deps": ["use"]
  }
, "all":
  { "type": "for"
  , "var": ["whom"]
  , "values": ["Munich", "Brussels"]
  , "dep": ["postprocessed"]
  }
, "main":
  { "type": "generic"
  , "outs": ["main.out.txt"]
  , "cmds":
    ["cat Munich/postprocessed.txt Brussels/postprocessed.txt > main.out.txt"]
  , "deps": ["all"]
  }
}
$
```

Just example

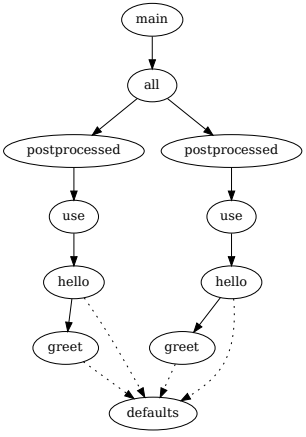
Example application, built with just (configured-target graph)

```
$ cat TARGETS
{ "hello":
  { "type": ["@", "rules", "CC", "binary"]
  , "name": ["hello"]
  , "srcs": ["hello.cpp"]
  , "private-deps": ["greet"]
  }
, "greet":
  { "type": ["@", "rules", "CC", "library"]
  , "arguments_config": ["whom"]
  , "name": ["greet"]
  , "hdrs": ["greet.hpp"]
  , "srcs": ["greet.cpp"]
  , "private-defines":
    [ { "type": "join"
      , "$1":
        ["WHOM=\"", {"type": "var", "name": "whom", "default": "World"}, "\""]
      }
    ]
  }
, "use":
  { "type": "generic"
  , "outs": ["use.txt"]
  , "cmds": ["./hello > use.txt"]
  , "deps": ["hello"]
  }
, "postprocessed":
  { "type": "generic"
  , "outs": ["postprocessed.txt"]
  , "cmds": ["tr 'a-z' 'A-Z' < use.txt > postprocessed.txt"]
  , "deps": ["use"]
  }
, "all":
  { "type": "for"
  , "var": ["whom"]
  , "values": ["Munich", "Brussels"]
  , "dep": ["postprocessed"]
  }
, "main":
  { "type": "generic"
  , "outs": ["main.out.txt"]
  , "cmds":
    ["cat Munich/postprocessed.txt Brussels/postprocessed.txt > main.out.txt"]
  , "deps": ["all"]
  }
}
$
```

Just example

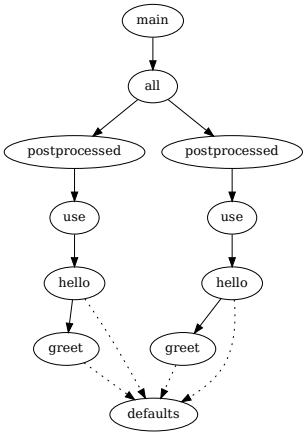
Example application, built with just (configured-target graph)

```
$ cat TARGETS
{ "hello":
  { "type": ["@", "rules", "CC", "binary"]
  , "name": ["hello"]
  , "srcs": ["hello.cpp"]
  , "private-deps": ["greet"]
  }
, "greet":
  { "type": ["@", "rules", "CC", "library"]
  , "arguments_config": ["whom"]
  , "name": ["greet"]
  , "hdrs": ["greet.hpp"]
  , "srcs": ["greet.cpp"]
  , "private-defines":
    [ { "type": "join"
      , "$1":
        ["WHOM=\"", {"type": "var", "name": "whom", "default": "World"}, "\""]
      }
    ]
  }
, "use":
  { "type": "generic"
  , "outs": ["use.txt"]
  , "cmds": ["./hello > use.txt"]
  , "deps": ["hello"]
  }
, "postprocessed":
  { "type": "generic"
  , "outs": ["postprocessed.txt"]
  , "cmds": ["tr 'a-z' 'A-Z' < use.txt > postprocessed.txt"]
  , "deps": ["use"]
  }
, "all":
  { "type": "for"
  , "var": ["whom"]
  , "values": ["Munich", "Brussels"]
  , "dep": ["postprocessed"]
  }
, "main":
  { "type": "generic"
  , "outs": ["main.out.txt"]
  , "cmds":
    ["cat Munich/postprocessed.txt Brussels/postprocessed.txt > main.out.txt"]
  , "deps": ["all"]
  }
}
$
```



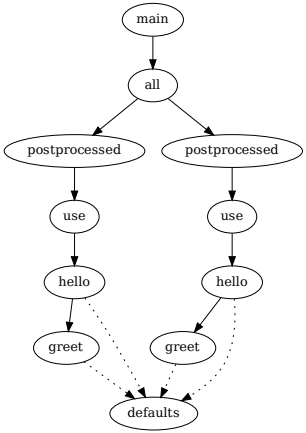
Just example

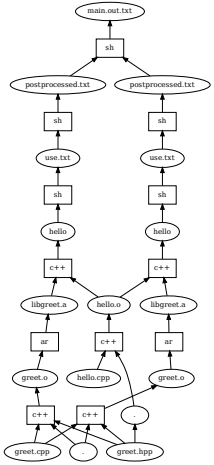
Example application, built with just (configured-target graph + action graph)



Just example

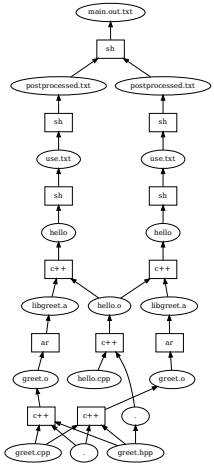
Example application, built with just (configured-target graph + action graph)





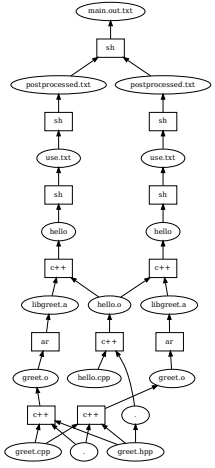
Just example

Example application, built with just (actual build)



Just example

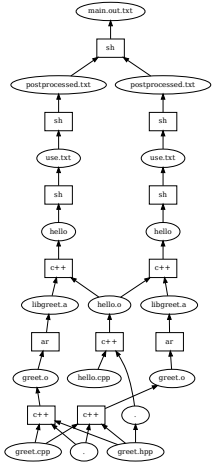
Example application, built with just (actual build)




```
$ just build -C repos.json main
INFO: Requested target is [["@","","","main"],{}]
INFO: Analysed target [["@","","","main"],{}]
INFO: Export targets found: 0 cached, 0 uncached, 0 not eligible for caching
INFO: Discovered 12 actions, 2 trees, 0 blobs
INFO: Building [["@","","","main"],{}].
INFO: Processed 12 actions, 0 cache hits.
INFO: Artifacts built, logical paths are:
        main.out.txt [72519212fd2388ceea246b0c536ff106047a6223:28:f]
$
```

Just example

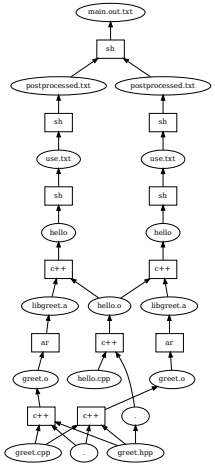
Example application, built with just (actual build)




```
$ just build -C repos.json main
INFO: Requested target is [["@","","","main"],{}]
INFO: Analysed target [["@","","","main"],{}]
INFO: Export targets found: 0 cached, 0 uncached, 0 not eligible for caching
INFO: Discovered 12 actions, 2 trees, 0 blobs
INFO: Building [["@","","main"],{}].
INFO: Processed 12 actions, 0 cache hits.
INFO: Artifacts built, logical paths are:
       main.out.txt [72519212fd2388ceea246b0c536ff106047a6223:28:f]
$

$ just install -C repos.json -o . main
INFO: Requested target is [["@","","","main"],{}]
INFO: Analysed target [["@","","","main"],{}]
INFO: Export targets found: 0 cached, 0 uncached, 0 not eligible for caching
INFO: Discovered 12 actions, 2 trees, 0 blobs
INFO: Building [["@","","main"],{}].
INFO: Processed 12 actions, 12 cache hits.
INFO: Artifacts can be found in:
       /worker/build/62ae6a5ffde7e151/root/work/example/main.out.txt [72519212fd2388ceea246b0c536f
$
```

Just example

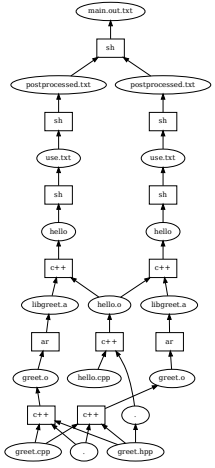
Example application, built with just (actual build)




```
$ just build -C repos.json main
INFO: Requested target is [["@","","","main"],{}]
INFO: Analysed target [["@","","","main"],{}]
INFO: Export targets found: 0 cached, 0 uncached, 0 not eligible for caching
INFO: Discovered 12 actions, 2 trees, 0 blobs
INFO: Building [["@","","","main"],{}].
INFO: Processed 12 actions, 0 cache hits.
INFO: Artifacts built, logical paths are:
       main.out.txt [72519212fd2388ceea246b0c536ff106047a6223:28:f]
$

$ just install -C repos.json -o . main
INFO: Requested target is [["@","","","main"],{}]
INFO: Analysed target [["@","","","main"],{}]
INFO: Export targets found: 0 cached, 0 uncached, 0 not eligible for caching
INFO: Discovered 12 actions, 2 trees, 0 blobs
INFO: Building [["@","","","main"],{}].
INFO: Processed 12 actions, 12 cache hits.
INFO: Artifacts can be found in:
       /worker/build/62ae6a5ffde7e151/root/work/example/main.out.txt [72519212fd2388ceea246b0c536f
$

$ cat main.out.txt
HELLO MUNICH
HELLO BRUSSELS
$
```

Just example

Example application, built with just (actual build)




```
$ just build -C repos.json main -P main.out.txt
INFO: Requested target is [["@","","","main"],{}]
INFO: Analysed target [["@","","","main"],{}]
INFO: Export targets found: 0 cached, 0 uncached, 0 not eligible for caching
INFO: Discovered 12 actions, 2 trees, 0 blobs
INFO: Building [["@","","","main"],{}].
INFO: Processed 12 actions, 12 cache hits.
INFO: Artifacts built, logical paths are:
        main.out.txt [72519212fd2388ceea246b0c536ff106047a6223:28:f]
HELLO MUNICH
HELLO BRUSSELS
$
```

Patching example

Logical in-place patching (multi-repo config)

```
$ cat repos.json
{ "main": ""
, "repositories":
  { "":
    { "workspace_root": ["file", "../third-party"]
    , "target_root": ["file", "."]
    , "bindings": {"rules": "rules", "patches": "patches"}
    }
  , "rules": {"workspace_root": ["file", "../rules"]}
  , "patches": {"workspace_root": ["file", "patches"]}
  }
}
$
```

Patching example

Logical in-place patching (multi-repo config)

```
$ cat repos.json
{ "main": ""
, "repositories":
  { "":
    { "workspace_root": ["file", "../third-party"]
    , "target_root": ["file", "."]
    , "bindings": {"rules": "rules", "patches": "patches"}
    }
  , "rules": {"workspace_root": ["file", "../rules"]}
  , "patches": {"workspace_root": ["file", "patches"]}
  }
}
$
```

```
$ ls ../third-party
greet.cpp
greet.hpp
hello.cpp
$
```

Patching example

Logical in-place patching (multi-repo config)

```
$ cat repos.json
{ "main": ""
, "repositories":
  { "":
    { "workspace_root": ["file", "../third-party"]
    , "target_root": ["file", "."]
    , "bindings": {"rules": "rules", "patches": "patches"}
    }
  , "rules": {"workspace_root": ["file", "../rules"]}
  , "patches": {"workspace_root": ["file", "patches"]}
  }
}
$
```

```
$ ls ../third-party
greet.cpp
greet.hpp
hello.cpp
$

$ ls patches
TARGETS
hello.diff
$
```

Patching example

Logical in-place patching (multi-repo config)

```
$ cat repos.json
{ "main": ""
, "repositories":
  { "":
    { "workspace_root": ["file", "../third-party"]
    , "target_root": ["file", "."]
    , "bindings": {"rules": "rules", "patches": "patches"}
    }
  , "rules": {"workspace_root": ["file", "../rules"]}
  , "patches": {"workspace_root": ["file", "patches"]}
  }
}
$
```

```
$ cat patches/hello.diff
--- hello.orig.cpp  2023-01-25 17:15:35.300389968 +0100
+++ hello.cpp   2023-01-25 17:15:46.312414032 +0100
@@ -1,5 +1,5 @@
 #include "greet.hpp"
 int main(int argc, char *argv[]) {
-  greet("Hello");
+  greet("Bonjour");
   return 0;
 }
$
```

```
$ ls ../third-party
greet.cpp
greet.hpp
hello.cpp
$

$ ls patches
TARGETS
hello.diff
$
```

Patching example

Logical in-place patching (multi-repo config)

```
$ cat repos.json
{ "main": ""
, "repositories":
  { "":
    { "workspace_root": ["file", "../third-party"]
    , "target_root": ["file", "."]
    , "bindings": {"rules": "rules", "patches": "patches"}
    }
  , "rules": {"workspace_root": ["file", "../rules"]}
  , "patches": {"workspace_root": ["file", "patches"]}
  }
}
$
```

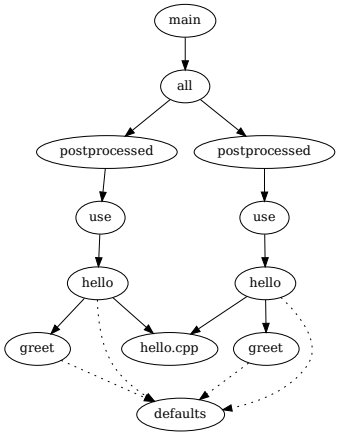
```
$ ls ../third-party
greet.cpp
greet.hpp
hello.cpp
$

$ ls patches
TARGETS
hello.diff
$
```

```
$ cat patches/hello.diff
--- hello.orig.cpp  2023-01-25 17:15:35.300389968 +0100
+++ hello.cpp   2023-01-25 17:15:46.312414032 +0100
@@ -1,5 +1,5 @@
 #include "greet.hpp"
 int main(int argc, char *argv[]) {
-  greet("Hello");
+  greet("Bonjour");
   return 0;
 }
$
```

```
--- TARGETS.orig
+++ TARGETS
@@ -42,4 +42,9 @@
     ["cat Munich/postprocessed.txt Brussels/postprocessed.txt > main.out.txt"]
   , "deps": ["all"]
   }
+, "hello.cpp":
+  { "type": ["@", "rules", "patch", "file"]
+  , "src": [["FILE", null, "hello.cpp"]]
+  , "patch": [["@", "patches", "", "hello.diff"]]
+  }
 }
```

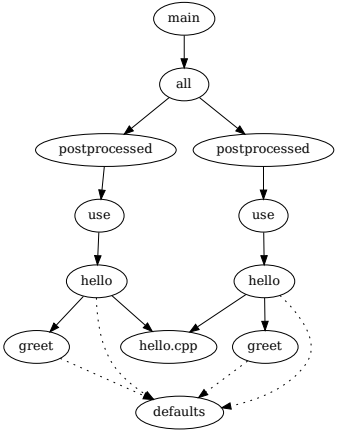
Patching example

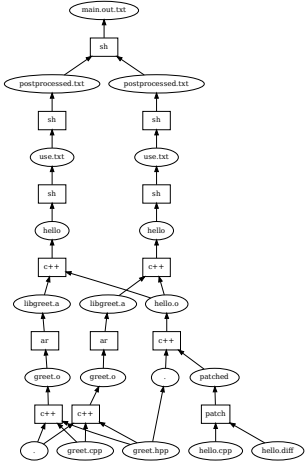
Logical in-place patching (target graph + action graph)



Patching example

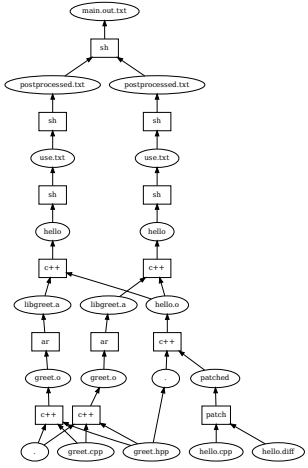
Logical in-place patching (target graph + action graph)





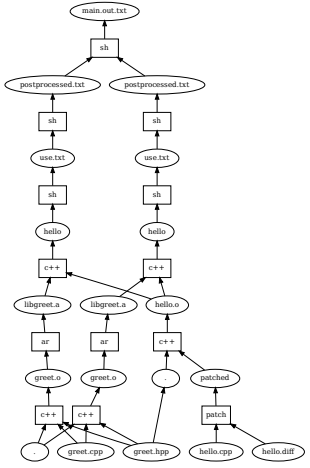
Patching example

Logical in-place patching (actual build)



Patching example

Logical in-place patching (actual build)




```
$ just build -C repos.json main -P main.out.txt
INFO: Requested target is [["@","","","main"],{}]
INFO: Analysed target [["@","","","main"],{}]
INFO: Export targets found: 0 cached, 0 uncached, 0 not eligible for caching
INFO: Discovered 13 actions, 2 trees, 1 blobs
INFO: Building [["@","","","main"],{}].
INFO: Processed 13 actions, 0 cache hits.
INFO: Artifacts built, logical paths are:
        main.out.txt [05e0946c10ca29543ef3e3a39c45daa4f0096507:32:f]
BONJOUR MUNICH
BONJOUR BRUSSELS
$
```

Summary

- Modern build systems should abandon the restriction to require a unique location for artifacts in the file system
- We propose to apply *staging* in current and emerging build systems
- Advantages of staging
  - No need to artificially invent new names to avoid conflicts
  - More readable and easier to understand
  - Better to maintain and more efficient to evaluate
  - Allows to use a single `isystem` include path to put required library header files
  - Seamless composition of multi-repo builds as each target has its own view of the world independent of the place of its definition

Sources

Our project

- `https://github.com/just-buildsystem/justbuild`
- License: Apache 2.0

Sources

Our project

- `https://github.com/just-buildsystem/justbuild`
- License: Apache 2.0

Now, the stage is yours!

Thanks for your attention!