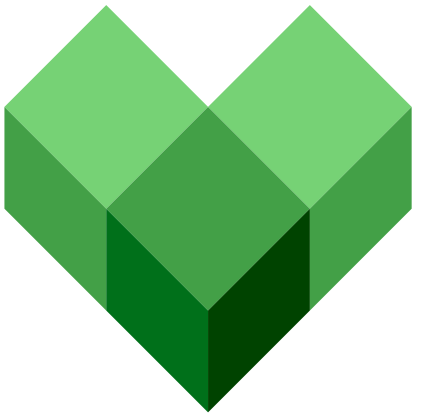
# Bazel and External Repositories

## Which version do you get?



Klaus Aehlig

October 9–10, 2018

# Imagine...

- You freshly check out your project.

# Imagine...

- You freshly check out your project.
- The `WORKSPACE` file describes the branches followed.

# Imagine...

- You freshly check out your project.
- The `WORKSPACE` file describes the branches followed.

```
load("@bazel_tools//tools/build_defs/repo:git.bzl",
     "git_repository")
...
git_repository(
  name = "com_google_protobuf",
  remote = "https://github.com/google/protobuf",
  branch = "master",
  patch_cmds = ["find .  -name '*.sh' -exec ..."],
)
...
```

# Imagine...

- You freshly check out your project.
- The `WORKSPACE` file describes the branches followed.

```
load("@bazel_tools//tools/build_defs/repo:git.bzl",
     "git_repository")
...
git_repository(
  name = "com_google_protobuf",
  remote = "https://github.com/google/protobuf",
  branch = "master",
  patch_cmds = ["find .  -name '*.sh' -exec ..."],
)
...
```

*Fully abstract description!*
*Only to be changed, when a new dependency is added.*

# Imagine...

- You freshly check out your project.
- The `WORKSPACE` file describes the branches followed.

# Imagine...

- You freshly check out your project.
- The `WORKSPACE` file describes the branches followed.
- `bazel build //...`

# Imagine. . .

- You freshly check out your project.
- The `WORKSPACE` file describes the branches followed.
- `bazel build //...`

  *. . . and you build at the latest known-good snapshot!*

# Imagine...

- You freshly check out your project.
- The `WORKSPACE` file describes the branches followed.
- `bazel build //...`

# Imagine...

- You freshly check out your project.
- The `WORKSPACE` file describes the branches followed.
- `bazel build //...` for an older version of your project

# Imagine. . .

- You freshly check out your project.
- The `WORKSPACE` file describes the branches followed.
- `bazel build //...` for an older version of your project

*. . . and you build against the snapshot used at that time!*

# Imagine...

- You freshly check out your project.
- The `WORKSPACE` file describes the branches followed.
- `bazel build //...`

# Imagine...

- You freshly check out your project.
- The `WORKSPACE` file describes the branches followed.
- `bazel build //...`
  - `WORKSPACE` file ignored

# Imagine...

- You freshly check out your project.
- The `WORKSPACE` file describes the branches followed.
- `bazel build //...`
  - `WORKSPACE` file ignored
  - `resolved.bzl` read instead *(generated, committed!)*

# Imagine...

- You freshly check out your project.
- The `WORKSPACE` file describes the branches followed.
- `bazel build //...`
  - `WORKSPACE` file ignored
  - `resolved.bzl` read instead *(generated, committed!)*
    - precise commit ids, instead of branches

# Imagine...

- You freshly check out your project.
- The `WORKSPACE` file describes the branches followed.
- `bazel build //...`
  - `WORKSPACE` file ignored
  - `resolved.bzl` read instead *(generated, committed!)*
    - precise commit ids, instead of branches
    - hashes of the generated directory

      $\rightsquigarrow$ *definitely the same code, even with transformations!*

# Imagine. . .

- You freshly check out your project.
- The `WORKSPACE` file describes the branches followed.
- `bazel build //...`
  - `WORKSPACE` file ignored
  - `resolved.bzl` read instead *(generated, committed!)*
    - precise commit ids, instead of branches
    - hashes of the generated directory
      $\rightsquigarrow$ *definitely the same code, even with transformations!*
    - actually, just a Starlark value

# Imagine...

- You freshly check out your project.
- The `WORKSPACE` file describes the branches followed.
- `bazel build //...`
  - `WORKSPACE` file ignored
  - `resolved.bzl` read instead *(generated, committed!)*
    - precise commit ids, instead of branches
    - hashes of the generated directory
      $\rightsquigarrow$ *definitely the same code, even with transformations!*
    - actually, just a Starlark value $\rightsquigarrow$ *build can use it*

```
load("//:resolved.bzl", "resolved)

for wsentry in resolved:
  repo = wsentry["original_attributes"]["name"]
  for actual in wsentry["repositories"]:
    ...
```

# Imagine...

- You freshly check out your project.
- The `WORKSPACE` file describes the branches followed.
- `bazel build //...` *(correct snapshot)*

# Imagine...

- You freshly check out your project.
- The `WORKSPACE` file describes the branches followed.
- `bazel build //...` *(correct snapshot)*
- *update dependency snapshot:* `bazel sync`

# Imagine...

- You freshly check out your project.
- The `WORKSPACE` file describes the branches followed.
- `bazel build //...` *(correct snapshot)*
- *update dependency snapshot:* `bazel sync`

```
$ bazel sync
...
INFO: Repository rule 'com_google_protobuf'
returned:  {"commit":  "c27d6a56...", ...}
...
$
```

# Imagine...

- You freshly check out your project.
- The `WORKSPACE` file describes the branches followed.
- `bazel build //...` *(correct snapshot)*
- *update dependency snapshot:* `bazel sync`

```
$ bazel sync
...
INFO: Repository rule 'com_google_protobuf'
returned:  {"commit":  "c27d6a56...", ...}
...
$
```

  - `WORKSPACE` file fully executed, unconditionally.

# Imagine...

- You freshly check out your project.
- The `WORKSPACE` file describes the branches followed.
- `bazel build //...` *(correct snapshot)*
- *update dependency snapshot:* `bazel sync`

```
$ bazel sync
...
INFO: Repository rule 'com_google_protobuf'
returned: {"commit": "c27d6a56...", ...}
...
$
```

  - `WORKSPACE` file fully executed, unconditionally.
  - versions (and hashes!) recorded in `resolved.bzl`

# Imagine...

- You freshly check out your project.
- The `WORKSPACE` file describes the branches followed.
- `bazel build //...` *(correct snapshot)*
- *update dependency snapshot:* `bazel sync`

```
$ bazel sync
...
INFO: Repository rule 'com_google_protobuf'
returned: {"commit": "c27d6a56...", ...}
...
$
```

  - `WORKSPACE` file fully executed, unconditionally.
  - versions (and hashes!) recorded in `resolved.bzl`
  - meaningful diff

# This is reality!

(as of Bazel 0.19; get the latest rc now)

# This is reality!

(as of Bazel 0.19; get the latest rc now)

Added as an experimental opt-in, controlled by `.bazelrc`

# This is reality!

(as of Bazel 0.19; get the latest rc now)

Added as an experimental opt-in, controlled by `.bazelrc`

```
sync --experimental_repository_resolved_file=resolved.bzl
build --experimental_resolved_file_instead_of_workspace=resolved.bzl
build --experimental_repository_hash_file=resolved.bzl
build --experimental_verify_repository_rules=...
```

# The Future

We have many ideas for the future. . .

- more rules to return versions (besides `git_repository`)
- meta-rules *(“These packages and their dependencies”)*
- source-like vs configure-like rules
- enable `resolved.bzl` by default
- dependency discovery and `install` targets (`autotools`-like)
  *Probably can be done in Starlark right now.*
- . . .

. . . but we need your input to decide what is important. Talk to us!