The `name=` field for targets

Eric Arellano, May 14 2022

See Address-less target generators for the original proposal. This is a refresh.

Update May 27, 2022: Removed proposal to at first always require `name=`.

Background

Addresses

An Address is how Pants uniquely refers to targets: its units of build.

We have a shortcut for leaving off `name=`, described more below. When the shortcut was introduced in Pants v1, we only had the two "normal" syntax for addresses. Now, we have 16 permutations:

- Normal
  - `path/to`
  - `path/to:tgt`
- Generated targets
  - File-less
    - `path/to#gen`
    - `path/to:tgt#gen`
  - File
    - `path/to/file.ext`
    - `path/to/file.ext:tgt`
    - `path/to/file.ext:../to` (i.e. the default name)
    - `path/to/file.ext:../tgt` (i.e. explicit name)
- Parametrization:
  - `path/to@key=v`
  - `path/to:tgt@key=v`
  - `path/to#gen@key=v`
  - `path/to:tgt#gen@key=v`
  - `path/to/file.ext@key=v`
  - `path/to/file.ext:tgt@key=v`
  - `path/to/file.ext:../to@key=v` (i.e. the default name)
  - `path/to/file.ext:../tgt@key=v` (i.e. explicit name)

Problem #1: `name=` default is confusing

If you do not set `name=`, we default it to the directory name, e.g. `project:project`.

Not *consistent*.

- Doubles the number of permutations for Address syntax, from 8 to 16 possibilities!
- Result: Harder for users to understand and to know how to form addresses.

Not *intuitive*.

- You must have prior knowledge of this shortcut.
- Users often expect `dependencies=["path/to/dir"]` to depend on the whole directory, not a target that leaves off `name=`
    - Note that the default target could have any target type, e.g. `pex_binary`. This has confused users who didn't realize they were depending on a binary.

Problem #2: addresses not intuitive for generated targets

Even if we get rid of the default `name=` shortcut, the address for generated targets would still be more complex than desirable:

- File-less: `path/to:tgt#gen`
- File
    - `path/to/file.ext:tgt`
    - `path/to/file.ext:../tgt`

This requires that you know the name of the `:tgt` generator, which is not intuitive. For file targets, you must know whether ** source globs were used or not.

Instead, we want something simple like:

- `path/to#generated`
- `path/to/f.ext`
    - Regardless of if the target generator is in a parent directory! No `path/to/f.ext:../tgt`

Proposal: target generators sometimes don't need `name=`

Atom targets like `python_source` and `pex_binary` must always set `name=`, but target generators like `python_sources` and `python_requirements` can leave it off.

If a target generator does not have a name, it is not addressable and will not show up in goals like `list`. It also cannot be used as an alias for all of its generated targets.

Generated targets will have addresses like this when `name=` is left off:

- `path/to#generated`
- `path/to/f.ext`

There are two times when you must set `name=` on a target generator:

- Ambiguity in the generated target because 2+ generators use that entity.
    - Pants will error in this case.
    - We need to decide if *all* targets need a `name=`, vs. all but one.
- If you want an alias to all the generated targets.

UX wins

In general, we only need to teach 6 address syntax variants—regardless of using 1:1:1—instead of 16 w/ the status quo and 8 w/ the always-name proposal:

1. normal address, `path/to:tgt`
2. literal file path, `path/to/f.ext`
3. generator syntax, `path/to#generated`
4. parametrization:
    a. `path/to:tgt@key=v`
    b. `path/to/f.ext@key=v`
    c. `path/to#generated@key=v`

These variants are also better:

- More intuitive - no need to guess a target name.
- Less noisy.

Because target generators no longer work as aliases by default, we encourage using precise file-level targets. We want users to have `dependencies` be as precise as possible, so that is likely a good nudge.

UX concerns

The expectation is that 95% of the time, you can use target generators without a `name`. Arguably, the awkwardness of the 5% is offset by the UX wins for the 95%.

Sample data of # owners for first-party files:

Repo	1 owner	2 owners	>2 owners	# owned files
pantsbuild/pants	95.9%	3.1%	1%	1227
sureshjoshi/pants-plugins	89.2%	8.1%	2.7%	37
sureshjoshi/private	78.0%	22.0%	0.0%	381
jcannon/private	86.2%	4.9%	8.9%	2508
toolchainlabs/private	94.3%	5.7%	0%	2065
alonso/private	99.1%	0.9%	0%	2647
jonas/private	95.9%	3.8%	0.2%	442

Adding a new target generator can break existing targets.

- This depends if our ambiguity rules require *all* ambiguous target generators to have a `name=`, vs all but one.
- If we require *all* ambiguous target generators to have `name=`...you must now update all references to the original, now-ambiguous generated targets.
- Example:
    - You use `python_sources(sources=["**/*.py"])` over 100 files. For one file, you need to add a second target generator. If our ambiguity rules require a `name=` for both target generators, then now you changed the address for 99 unrelated files.
- There is a workaround—regardless of ambiguity rules—to create an atom target when you need a second copy of metadata, rather than a target generator.

Not clear to users where metadata is set.

- Example:
    - You have two `python_requirements()` target generators in the same directory. Their requirements.txt files are disjoint, so no ambiguity. Where does `3rdparty#django` come from?
    - Where does `src/py/project/dir/util.py` come from? It could be any ancestor directory.
- Our error message for invalid generated targets can no longer confidently say what all the actually generated targets are.
    - Currently, we say "//:reqs" does not generate `#bad` - did you mean one of …?

    - Because N target generators are now candidates, vs only 1, we probably don't want to list each generated target per N generators?
- We probably need mechanisms to answer where a generated target comes from:
    - E.g. `./pants peek` tells you the BUILD file (and line number?)
    - Error messages mention the BUILD file

Users having to learn special cases.

- We will have 12 address variants, vs. 16 currently and 8 with the always-name proposal:
    - Normal: `path/to:tgt`
    - Generated targets
        - File-less
            - `path/to#gen`
            - `path/to:tgt#gen`
        - File
            - `path/to/file.ext`
            - `path/to/file.ext:tgt`
            - `path/to/file.ext:../tgt`
    - Parametrization:
        - `path/to:tgt@key=v`
        - `path/to#gen@key=v`
        - `path/to:tgt#gen@key=v`
        - `path/to/file.ext@key=v`
        - `path/to/file.ext:tgt@key=v`
        - `path/to/file.ext:../tgt@key=v`.

Technical feasibility

This is going to be tough to implement, but I think it's feasible.

- I confirmed the foundational rule `Address -> WrappedTarget` can add `Get(AddressFamilies)` to its rule w/o rule graph issues. What that means is if we have a generated target address, we will look up *all* ascendent target generators and see if any of them generate the target. Right now, we only look up the one dedicated target generator.
- Our BUILD file parsing types—`TargetAdaptor` and `AddressFamily`—need to be redesigned.
    - We no longer have a way to uniquely identify each BUILD file entry, vs. right now we use the `name` as a mapping.
- We use `.maybe_convert_to_target_generator()` in several places, e.g. disambiguating dependency inference. That must now be made fallible.
- The only place we directly operate on a target generator in Build rules is `go_mod`. I think I have a workaround.
    - Generally, it's not safe for build rules to rely on target generators anymore.

Deprecation plan

Problem

The new semantics cannot coexist with the status quo default of `name=`, due to these breaking changes:

- Using the default `name=` for a non-target generator.
- All current references to target generators without a `name=` will become errors.
- All current references to target generators with a) an explicit `name=` set to the directory's name, and b) without the `:tgt` portion, will become errors.
- >1 target generator for the same entity requires that every target generator has an explicit name.
    - Even if we decide to allow one to leave it off, we still need to check for 3+ target generators.
- Explicit references to generated file targets from a subdirectory will need an explicit `name=`.
    - Example: `path/to/project/f.ext:../to` would become `path/to/project/f.ext`. So the user would need to set a `name=` in the transition time.

Once we ban these five cases, we will be safe to remove the status quo semantics and turn on the new semantics.

Plan

The five ambiguous cases above will be deprecated. We still leave off `name=` for most target generators with `tailor`.

Implementation plan:

1. Record with `Target` when `name=` was explicitly left off.
2. Stop rendering target generators with the `name=` left off, e.g. in `list`.
3. Deprecate the above cases & add fixers with `update-build-files`.
    a. The fix is to add `name=` to generators which are triggering the above 5 ambiguous cases.
    b. We must both fix target declarations and references (e.g. `dependencies` field)
    c. For the rglobs case, do we want to ban not setting the `name` entirely? Or only if you refer to a generated file from a subdirectory?
4. After N releases, convert warnings to errors.

How to document during the transition period:

- If a target generator has `name=` set, then you can refer to it as an alias.
- If `name=` is left off, using ** sources is not safe.
    - Technically, some cases are safe. But too hard to explain the special cases.
- Only target generators can leave off `name=`

Originally, I proposed always requiring `name=` while we remove the old semantics. While easier to implement and document, it would result in too much `BUILD` file churn.

Q&A etc

Thread w/ Benjy if we need to rethink targets more profoundly first .

If the proposal ends up being technically infeasable, would our deprecation plan have made things better than before?