Pants 1.0 Release Plan

Summary

This lists the features we need to get to the Pants 1.0 release and policies we will put in place to maintain the stability of release versions of Pants.

It additionally covers the 1.0.0 announcement, and any coordinated efforts lead by contributing organizations to assist in the announcement.

Announcement

1.0.0 is a significant symbolic milestone, and we should attempt to make a bit of a splash by announcing it.

Open questions

1. Date?
2. Publicity?
   a. Blog posts?
3. Message?

    a. example: “Pants is ready for ‘end-users’!“
4. Celebration?
    a. Party/Meetup in SF?

Blockers

Goals

These should all have corresponding issues in the issue tracker labeled 1.0.0

- Pants can be built with an empty pants.ini [2534]
- Identify API modules and methods that Plugins can rely on [2710]
- Website supports documentation for multiple release versions plus a development version [2639]
- Make sure that nailgun and ivy are not needed for a python-only repo [867, 940]
- Better test coverage codifying some of the decisions laid out by this document.
    - Insuring modules in non ‘private’ or ‘exp’ directories are marked with an API tag
    - Options may not be not removed or renamed (for API and non API modules)
    - API modules may not be removed or renamed
    - Methods in API public modules cannot be removed or renamed
- Deprecated changes to the API will be supported for 2 minor release versions.

Non Goals

- Pants installs and runs in a python 3 environment, and this is always backstopped by CI

Documentation

Currently we publish docs to pantsbuild.gitub.io by running a script that rewrites the data for the entire web site on every publish. Will need evolve this to a place where we want to support both release and development documentation. See [2639]

- **Create a separate main website from the generated code.**
    - **We could just make this a new github repo pantsbuild/website-main to hold** the source for the top level pants website. The rationale for this is that It needs to be built and published independently of changes to the 1.x and future releases. We can juggle this around and make the rest of the site pretty and use whatever formats we want
    - *OR* use a CMS like Drupal. This will allow us to “easily” edit the top level content and use some kind of canned theme that looks good.

        - Corporate Clean
        - Likable
        - Responsive Blog
- **Create a new top level landing page.** This page will contain all meta information about the project:
    - Who uses pants
        - Powered by Pants
    - What is pants
        - Conceptual overview
        - Links to presentations, white papers, and other resources
    - Why use pants
    - News
    - Project governance
        - How to Contribute
        - How to Ask
        - Contact information
    - Release Documentation: links to the most current version of each release
    - About the documentation
    - Credits

- **Move existing website generation to publish under a /releases/ directory**.
    - publish_docs.sh inside of each release will be nailed to publishing under /releases/<version>/
    - The release will pull in a common heading from the upper directory so that users can navigate between releases.
    - The version number will be updated for minor releases only.
        - 1.0 gets a release page and all minor updates overwrite the 1.0 release page.
        - When 1.1 is released, we change the version number in publish_docs.sh so that a new page is generated.
- **Refactor the generated release documentation.**
    - Remove the meta information that has migrated to the top level landing page.
- **Move our top level website to pantsbuild.io or pantsbuild.org.**
    - Makes us look like a ‘real’ project
    - bazel.io buckbuild.com already do it
    - Use some hosting provider other than github to host our CMS
    - Figure out a way to pay for it. If we can charge it to a credit card, Square can pay.

API change policy post 1.0

This section is obsolete. See http://pantsbuild.github.io/deprecation_policy.html

These rules are in effect for release branches until the next 2 minor releases (e.g. if the feature is available 1.0.x is should continue to be available in 1.1.x and 1.2.x and can be removed in 1.3.x). This assumes a rough timeline of 3 months lifetime per minor release.

API Definition

This policy applies to:

- Modules under src/main/python/pants that do not start with an _
- Modules under src/test/python/pants_test that are marked *API:Public* in pydoc

Excluding:

- Modules under `src/python/pants` a directory named '**exp**' or prefixed with '**_**'
- Module under `tests/python/pants_test` not marked *API:Public*
- Any module prefixed with '_'
- Any method prefixed with '_'
- Any method prefixed with '`private_`'
- Modules under any other directory including `contrib, examples, testprojects`

Allowed API changes:

- Adding a new module
- Adding new command line options
- Adding new features to existing modules
- Deprecate and warn about an API that has been refactored
- Deprecate and warn about an option that has been refactored
- Adding new named parameters to a public API method
- Adding/removing/renaming any module or method in a directory named **'exp'** or starting with the prefix **'_'**
- Adding/removing/renaming any module prefixed with **'_'**
- Adding/removing/renaming any method prefixed with **'_'** or **'private_'**
- Fixing bugs
    - Caveat, sometimes builds rely on buggy behavior
    - Be more precise as to what a bug actually means and consider applying fixes on a case-by-case basis.

Disallowed API changes:

- Deprecated options must continue to work as before
- Existing API modules cannot be moved.
- Options cannot be removed

- Parameters cannot be removed from API methods (any public method in an API module)
- Changing the behavior of a method that breaks existing assumptions
    - e.g. changing a method that used to do transitive resolution to intransitive resolution would be disallowed, but adding a new named parameter to change the behavior would be allowed.
- Changes that introduce significant performance regressions by default
    - A significant regression would be a slowdown of > 10%
    - If a new feature is needed that would slow down performance more than 10%, it should be put behind an option

Identifying API modules and methods

(searching for good ideas here…)

Running this in each of our repos will tell us what that repo imports:

```
git grep -E '^from pants\..* import .*$' | cut -d: -f2 | sort | uniq
```

[Add your results to the Appendix below. Multiline import statements will require manual addition]

File by File Analysis for API:

This section is obsolete. See http://pantsbuild.github.io/deprecation_policy.html

Use this spreadsheet to make sure we analyze each file:

Modules under `src/python/pants`:

- Privatize modules that are not for use in plugins by renaming them and prefixing them with '_'
- Privatize packages by renaming the directory and prefixing it with '_'
- Privatize methods not intended to be exposed outside of the module by renaming them and prefixing them with '_'
- Privatize methods used between pants modules by prefixing them with '**`private_`**'

Under `tests/python/pants_test`:

- The pydoc for public modules should contain:

```
API:Public Usable by plugins. Should remain stable for major releases
```

- The package directory for test modules should be the same as the primary modules they are testing

Discussion about using docstrings to indicate public/private:

The primary concern is that renaming all method's/modules with _ will add a large amount of noise since most api's are intended to be private. The proposal document APIs as being public or private, and that we should spend all of this effort writing tests to enforce these requirements.

https://pantsbuild.slack.com/archives/general/p1454365812001067

Proposal:

Use the following syntax inside of a docstring to indicate an api is public. The absence of this marker in the docstring would indicate private, which is intended to be the majority of cases. It may also be useful to register public api's somewhere so that we can generate documentation based on docstrings. This is currently done with BUILD dictionary but relies on a call to: `self.context.build_file_parser.registered_aliases()`

```
:API: public
```

An example of using docstrings to mark API's can be found at:
https://rbcommons.com/s/twitter/r/3417/

Appendix 1: Existing references to API methods

Square's internal repo:

```
from pants.backend.codegen.targets.java_wire_library import JavaWireLibrary
from pants.backend.codegen.tasks.simple_codegen_task import SimpleCodegenTask
from pants.backend.jvm.ivy_utils import IvyUtils, IvyInfo, IvyModuleRef
from pants.backend.jvm.jar_dependency_utils import M2Coordinate
from pants.backend.jvm.repository import Repository
from pants.backend.jvm.subsystems.jvm import JVM
```

```
from pants.backend.jvm.targets.exclude import Exclude
from pants.backend.jvm.targets.exportable_jvm_library import ExportableJvmLibrary
from pants.backend.jvm.targets.jar_dependency import JarDependency
from pants.backend.jvm.targets.jar_library import JarLibrary
from pants.backend.jvm.targets.java_library import JavaLibrary
from pants.backend.jvm.targets.jvm_binary import JvmBinary
from pants.backend.jvm.targets.jvm_prep_command import JvmPrepCommand
from pants.backend.jvm.targets.unpacked_jars import UnpackedJars
from pants.backend.jvm.tasks.checkstyle import Checkstyle
from pants.backend.jvm.tasks.classpath_products import ArtifactClasspathEntry
from pants.backend.jvm.tasks.classpath_products import ClasspathProducts
from pants.backend.jvm.tasks.classpath_products import ClasspathProducts,
ArtifactClasspathEntry
from pants.backend.jvm.tasks.ivy_task_mixin import IvyTaskMixin
from pants.backend.jvm.tasks.jvm_tool_task_mixin import JvmToolTaskMixin
from pants.backend.jvm.tasks.nailgun_task import NailgunTask
from pants.backend.jvm.tasks.run_jvm_prep_command import RunJvmPrepCommandBase
from pants.backend.jvm.tasks.unpack_jars import UnpackJars
from pants.backend.project_info.tasks.export import ExportTask
from pants.backend.project_info.tasks.idea_gen import IdeaGen
from pants.base.build_environment import get_buildroot
from pants.base.exceptions import TargetDefinitionException
from pants.base.exceptions import TaskError
from pants.base.generator import Generator, TemplateData
from pants.base.payload import Payload
from pants.base.payload_field import JarsField
from pants.base.payload_field import PayloadField, PrimitiveField, stable_json_sha1
from pants.base.payload_field import PrimitiveField
from pants.base.revision import Revision
from pants.base.validation import assert_list
from pants.base.workunit import WorkUnit, WorkUnitLabel
from pants.base.workunit import WorkUnitLabel
from pants.binaries import binary_util
from pants.build_graph.address import Address
from pants.build_graph.build_file_aliases import BuildFileAliases
from pants.build_graph.resources import Resources
from pants.build_graph.target import Target
from pants.fs.archive import ZIP
from pants.goal.goal import Goal
from pants.goal.task_registrar import TaskRegistrar as task
from pants.ivy.ivy_subsystem import IvySubsystem
from pants.java.distribution.distribution import DistributionLocator
from pants.java.executor import SubprocessExecutor
from pants.option.config import Config
from pants.option.custom_types import dict_option
from pants.scm.git import Git
from pants.task.repl_task_mixin import ReplTaskMixin
from pants.task.task import Task
```

```
from pants.util.contextutil import temporary_dir
from pants.util.dirutil import safe_mkdir
from pants.util.dirutil import safe_mkdir, safe_mkdtemp
from pants.util.dirutil import safe_mkdir, safe_walk
from pants.util.dirutil import safe_mkdtemp
from pants.util.dirutil import touch
from pants.util.memo import memoized
from pants.util.memo import memoized_method, memoized_property
from pants.util.osutil import get_os_name, known_os_names, normalize_os_name,
OS_ALIASES
from pants.util.osutil import get_os_name, normalize_os_name
```