The Trac Ticket Workflow System

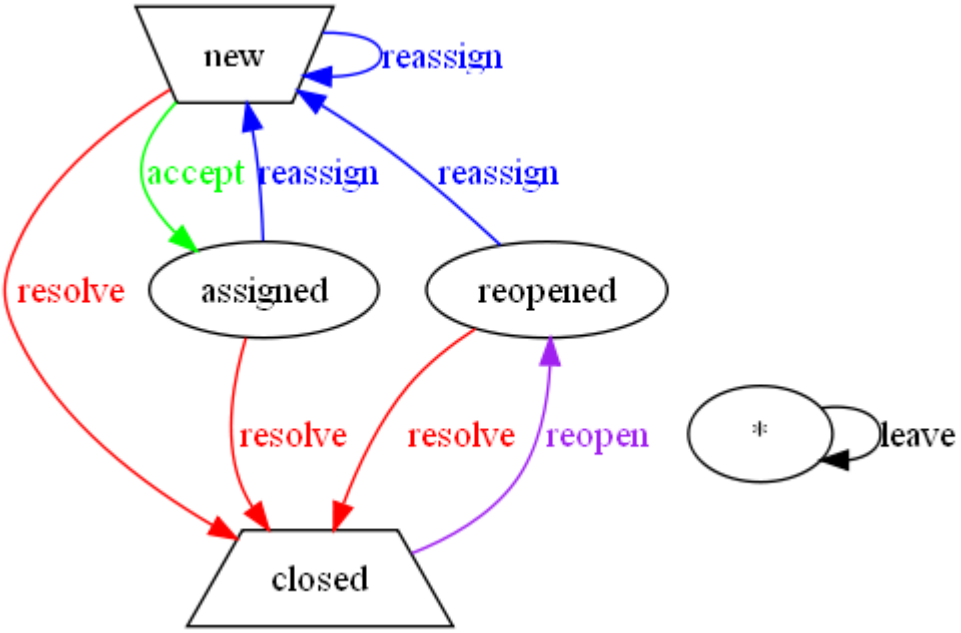
The Trac issue database provides a configurable workflow.

The Default Ticket Workflow

Environments upgraded from 0.10

When you run `trac-admin <env> upgrade`, your `trac.ini` will be modified to include a `[ticket-workflow]` section. The workflow configured in this case is the original workflow, so that ticket actions will behave like they did in 0.10.

Graphically, that looks like this:

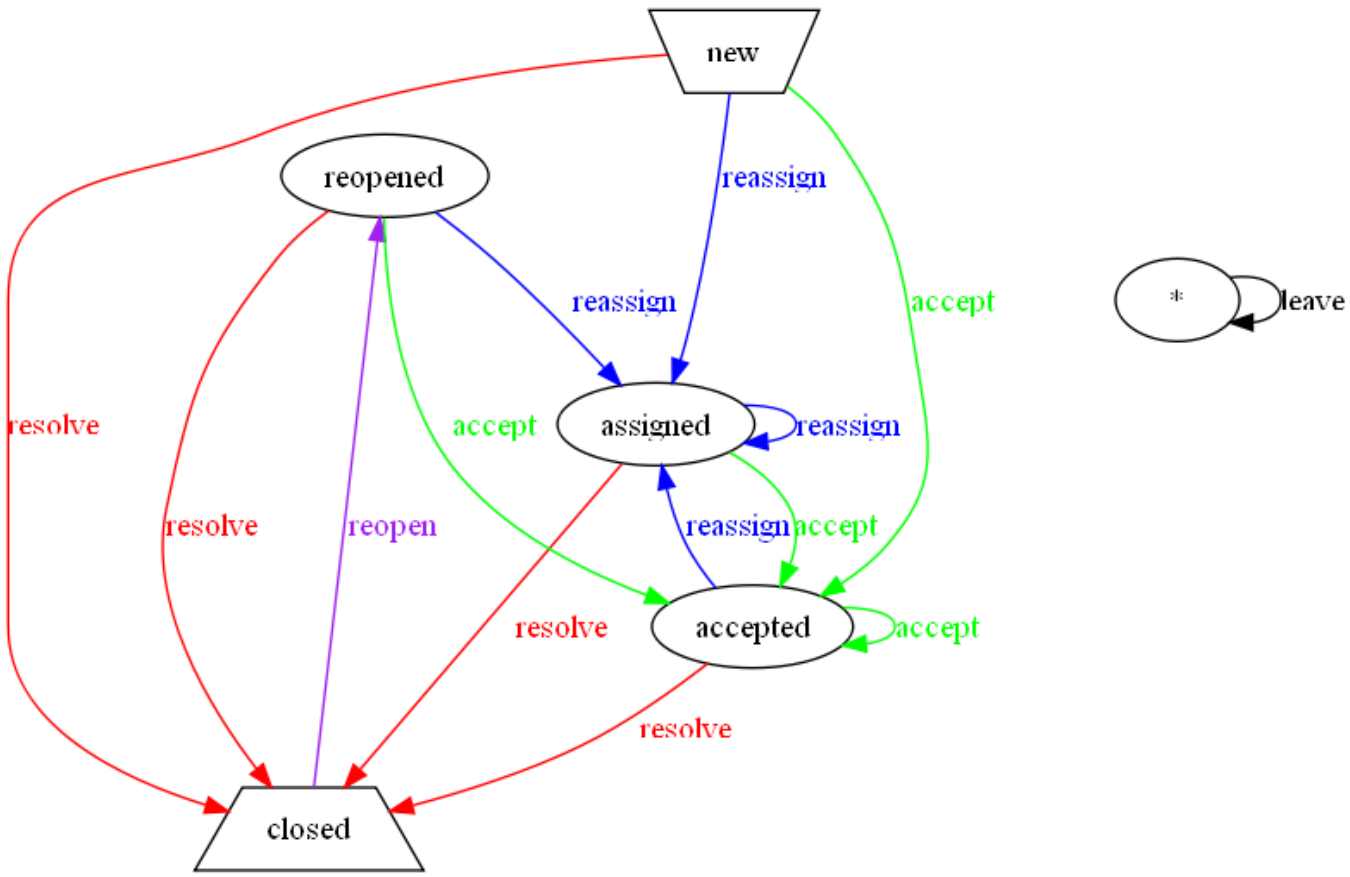


There are some significant "warts" in this; such as accepting a ticket sets it to 'assigned' state, and assigning a ticket sets it to 'new' state. Perfectly obvious, right? So you will probably want to migrate to "basic" workflow; contrib/workflow/migrate_original_to_basic.py may be helpful.

Environments created with 0.11

When a new environment is created, a default workflow is configured in your trac.ini. This workflow is the basic workflow (described in `basic-workflow.ini`), which is somewhat different from the workflow of the 0.10 releases.

Graphically, it looks like this:



Additional Ticket Workflows

There are several example workflows provided in the Trac source tree; look in contrib/workflow for `.ini` config sections. One of those may be a good match for what you want. They can be pasted into the `[ticket-workflow]` section of your `trac.ini` file. However if you have existing tickets then there may be issues if those tickets have states that are not in the new workflow.

Here are some diagrams of the above examples.

Basic Ticket Workflow Customization

Note: Ticket "statuses" or "states" are not separately defined. The states a ticket can be in are automatically generated by the transitions defined in a workflow. Therefore, creating a new ticket state simply requires defining a state transition in the workflow that starts or ends with that state.

Create a `[ticket-workflow]` section in `trac.ini`. Within this section, each entry is an action that may be taken on a ticket. For example, consider the `accept` action from `simple-workflow.ini`:

```
accept = new,accepted -> accepted
accept.permissions = TICKET_MODIFY
accept.operations = set_owner_to_self
```

The first line in this example defines the `accept` action, along with the states the action is valid in (`new` and `accepted`), and the new state of the ticket when the action is taken (`accepted`). The `accept.permissions` line specifies what permissions the user must have to use this action. The `accept.operations` line specifies changes that will be made to the ticket in addition to the status change when this action is taken. In this case, when a user clicks on `accept`, the ticket owner field is updated to the logged in user. Multiple operations may be specified in a comma separated list.

The available operations are:

- del_owner -- Clear the owner field.
- set_owner -- Sets the owner to the selected or entered owner.
  - *actionname*`.set_owner` may optionally be set to a comma delimited list or a single value.
- set_owner_to_self -- Sets the owner to the logged in user.

- del_resolution -- Clears the resolution field
- set_resolution -- Sets the resolution to the selected value.

  actionname`.set_resolution` may optionally be set to a comma delimited list or a single value.

```
Example:

resolve_new = new -> closed
resolve_new.name = resolve
resolve_new.operations = set_resolution
resolve_new.permissions = TICKET_MODIFY
resolve_new.set_resolution = invalid,wontfix
```

- leave_status -- Displays "leave as <current status>" and makes no change to the ticket.

Note: Specifying conflicting operations (such as `set_owner` and `del_owner`) has unspecified results.

```
resolve_accepted = accepted -> closed
resolve_accepted.name = resolve
resolve_accepted.permissions = TICKET_MODIFY
resolve_accepted.operations = set_resolution
```

In this example, we see the `.name` attribute used. The action here is `resolve_accepted`, but it will be presented to the user as `resolve`.

For actions that should be available in all states, `*` may be used in place of the state. The obvious example is the `leave` action:

```
leave = * -> *
leave.operations = leave_status
leave.default = 1
```

This also shows the use of the `.default` attribute. This value is expected to be an integer, and the order in which the actions are displayed is determined by this value. The action with the highest `.default` value is listed first, and is selected by default. The rest of the actions are listed in order of decreasing `.default` values. If not specified for an action, `.default` is 0. The value may be negative.

There are a couple of hard-coded constraints to the workflow. In particular, tickets are created with status `new`, and tickets are expected to have a `closed` state. Further, the default reports/queries treat any state other than `closed` as an open state.

While creating or modifying a ticket workfow, `contrib/workflow/workflow_parser.py` may be useful. It can create `.dot` files that GraphViz understands to provide a visual description of the workflow.

This can be done as follows (your install path may be different).

```
cd /var/local/trac_devel/contrib/workflow/
sudo ./showworkflow /srv/trac/PlannerSuite/conf/trac.ini
```

And then open up the resulting `trac.pdf` file created by the script (it will be in the same directory as the `trac.ini` file).

An online copy of the workflow parser is available at http://foss.wush.net/cgi-bin/visual-workflow.pl

After you have changed a workflow, you need to restart apache for the changes to take effect. This is important, because the changes will still show up when you run your script, but all the old workflow steps will still be there until the server is restarted.

Example: Adding optional Testing with Workflow

By adding the following to your [ticket-workflow] section of trac.ini you get optional testing. When the ticket is in new, accepted or needs_work status you can choose to submit it for testing. When it's in the testing status the user gets the option to reject it and send it back to needs_work, or pass the testing and send it along to closed. If they accept it then it gets automatically marked as closed and the resolution is set to fixed. Since all the old work flow remains, a ticket can skip this entire section.

```
testing = new,accepted,needs_work,assigned,reopened -> testing
testing.name = Submit to reporter for testing
```

```
testing.permissions = TICKET_MODIFY

reject = testing -> needs_work
reject.name = Failed testing, return to developer

pass = testing -> closed
pass.name = Passes Testing
pass.operations = set_resolution
pass.set_resolution = fixed
```

Example: Add simple optional generic review state

Sometimes Trac is used in situations where "testing" can mean different things to different people so you may want to create an optional workflow state that is between the default workflow's `assigned` and `closed` states, but does not impose implementation-specific details. The only new state you need to add for this is a `reviewing` state. A ticket may then be "submitted for review" from any state that it can be reassigned. If a review passes, you can re-use the `resolve` action to close the ticket, and if it fails you can re-use the `reassign` action to push it back into the normal workflow.

The new `reviewing` state along with its associated `review` action looks like this:

```
review = new,assigned,reopened -> reviewing
review.operations = set_owner
review.permissions = TICKET_MODIFY
```

Then, to integrate this with the default Trac 0.11 workflow, you also need to add the `reviewing` state to the `accept` and `resolve` actions, like so:

```
accept = new,reviewing -> assigned
[…]
resolve = new,assigned,reopened,reviewing -> closed
```

Optionally, you can also add a new action that allows you to change the ticket's owner without moving the ticket out of the `reviewing` state. This enables you to reassign review work without pushing the ticket back to the `new` status.

```
reassign_reviewing = reviewing -> *
reassign_reviewing.name = reassign review
reassign_reviewing.operations = set_owner
reassign_reviewing.permissions = TICKET_MODIFY
```

The full `[ticket-workflow]` configuration will thus look like this:

```
[ticket-workflow]
accept = new,reviewing -> assigned
accept.operations = set_owner_to_self
accept.permissions = TICKET_MODIFY
leave = * -> *
leave.default = 1
leave.operations = leave_status
reassign = new,assigned,reopened -> new
reassign.operations = set_owner
reassign.permissions = TICKET_MODIFY
reopen = closed -> reopened
reopen.operations = del_resolution
reopen.permissions = TICKET_CREATE
resolve = new,assigned,reopened,reviewing -> closed
resolve.operations = set_resolution
resolve.permissions = TICKET_MODIFY
review = new,assigned,reopened -> reviewing
review.operations = set_owner
review.permissions = TICKET_MODIFY
reassign_reviewing = reviewing -> *
```

```
reassign_reviewing.operations = set_owner
reassign_reviewing.name = reassign review
reassign_reviewing.permissions = TICKET_MODIFY
```

Example: Limit the resolution options for a new ticket

The above resolve_new operation allows you to set the possible resolutions for a new ticket. By modifying the existing resolve action and removing the new status from before the `->` we then get two resolve actions. One with limited resolutions for new tickets, and then the regular one once a ticket is accepted.

```
resolve_new = new -> closed
resolve_new.name = resolve
resolve_new.operations = set_resolution
resolve_new.permissions = TICKET_MODIFY
resolve_new.set_resolution = invalid,wontfix,duplicate

resolve = assigned,accepted,reopened -> closed
resolve.operations = set_resolution
resolve.permissions = TICKET_MODIFY
```

Advanced Ticket Workflow Customization

If the customization above is not extensive enough for your needs, you can extend the workflow using plugins. These plugins can provide additional operations for the workflow (like code_review), or implement side-effects for an action (such as triggering a build) that may not be merely simple state changes. Look at sample-plugins/workflow for a few simple examples to get started.

But if even that is not enough, you can disable the ConfigurableTicketWorkflow component and create a plugin that completely replaces it.

Adding Workflow States to Milestone Progress Bars

If you add additional states to your workflow, you may want to customize your milestone progress bars as well. See TracIni.

some ideas for next steps

New enhancement ideas for the workflow system should be filed as enhancement tickets against the `ticket system` component. If desired, add a single-line link to that ticket here. Also look at the [th:wiki:AdvancedTicketWorkflowPlugin] as it provides experimental operations.

If you have a response to the comments below, create an enhancement ticket, and replace the description below with a link to the ticket.

- the "operation" could be on the nodes, possible operations are:
  - **preops**: automatic, before entering the state/activity
  - **postops**: automatic, when leaving the state/activity
  - **actions**: can be chosen by the owner in the list at the bottom, and/or drop-down/pop-up together with the default actions of leaving the node on one of the arrows.

This appears to add complexity without adding functionality; please provide a detailed example where these additions allow something currently impossible to implement.

- operations could be anything: sum up the time used for the activity, or just write some statistical fields like

A workflow plugin can add an arbitrary workflow operation, so this is already possible.

- set_actor should be an operation allowing to set the owner, e.g. as a "preop":
  - either to a role, a person
  - entered fix at define time, or at run time, e.g. out of a field, or select.

This is either duplicating the existing `set_owner` operation, or needs to be clarified.

- Actions should be selectable based on the ticket type (different Workflows for different tickets)

Look into the [th:wiki:AdvancedTicketWorkflowPlugin]'s `triage` operation.