Black Belt Documentation

Release 0.16.0

Apiary

Sep 07, 2018

Contents

This is an internal tool for supporting development workflow inside Apiary. However, we decided to open-source it. Feel free to fork it and use it on your own or inside your company.

CHAPTER 1

Installation

On macOS, the easiest way to install blackbelt is via Homebrew:

```
$ brew install apiaryio/formulae/blackbelt
```

On other platforms you can install Blackbelt with `pip install blackbelt` providing you have Python and pip installed. Consult your platform for documentation on how to install Python and pip.

Note: Python 3.6+ is default now and Python 2.7 will be deprecated.

CHAPTER 2

Setup

With that, you should have `bb` command. Run interactive `bb init` and follow instructions. This is going to connect to services we are using in Apiary for futher interaction:

- GitHub
- Trello
- Slack

Retrieved tokens and configuration is stored in `~/.blackbelt`. Format is now just dumped JSON, don't rely on it; it's probably going to change in the future.

If you are using `bash`, you want to enable autocompletion. You can try it with:

```
eval "$( _BB_COMPLETE=source bb)"
```

and if it's working properly, put it into your `~/.bashrc`:

```
echo '_BB_COMPLETE=source bb > /tmp/_black_belt_autocompletion.sh'  >> ~/.bashrc
echo 'source /tmp/_black_belt_autocompletion.sh' >> ~/.bashrc
```

See click's documentation for more information.

CHAPTER 3

Upgrade

If you used Homebrew on macOS, use:

```
$ brew update
$ brew upgrade
```

Otherwise:

```
$ pip install -U blackbelt
```

If you get error along the lines of:

```
OSError: [Errno 1] Operation not permitted: '/tmp/pip-IYiPfC-uninstall/
```

you have a problem in your system installation (probably Mac OS X). You can either:

```
sudo pip install -U --ignore-installed blackbelt
```

or:

```
pip uninstall blackbelt
pip install blackbelt
```

CHAPTER 4

Structure

bb has subcommands that stand for an area of expertise. View help or command reference on how to use those.

This documents focus more on intended workflow from the perspective of two roles (those may be the same person, of course): Developer and Story Owner.

4.1 Developer

The *Developer* processes *Work Cards*, develops them and deploys them to production.

Those live on the *Work Board*.

Those are the task *Developer* may want to perform.

4.1.1 Development as usual

Open current task

Open current Trello card in browser:

```
bb t curcard
```

Move on to the next task

Move on to the next (Trello) card with:

```
bb t next
```

This:

1. Inspects `To Do` on the *Work Board* for the highest ticket assigned to you

2. **Creates a new local git branch inferred from ticket name and prefixed with your github prefix** (It is always forked from master and ensures master is up to date)
3. Moves the card to `Doing`
4. Opens the card in browser for review

Issuing Pull Request

`blackbelt.commands.gh.`**`pr_command`**(*card_url*)

Usage:

```
bb gh pr [CARD_URL]
```

See `threading` for more details.

Checking Dependencies

`blackbelt.commands.dep.`**`check`**()

4.1.2 Code review

Code review ensures the quality of the code and disperses the knowledge about the code and features through the team.

`blackbelt.commands.gh.`**`status_command`**(*pr_url*, *branch*)

Usage:

```
bb gh status https://github.com/apiaryio/apiary/pull/1234
bb gh status my_branch_name

Applicable for PRs and branches
```

`blackbelt.handle_github.`**`check_status`**(*pr_url=None*, *branch_name=None*, *error_on_failure=False*)

Returns status and required checks status of a given PR or branch.

This can be used to determine if a PR/branch can be merged without issues. Required checks might involve a status of a CI build, status of code reviews, etc. (see https://help.github.com/articles/about-required-status-checks/)

Pull requests

1. Checks for Pull Request current state (open/closed)
2. Retrieves required checks status (success/failure/pending)

Branches

1. Retrieves required checks status (success/failure/pending)

Merging Pull Request

`blackbelt.commands.gh.`**`merge_command`**(*pr_url*)

Usage:

```
bb gh merge https://github.com/apiaryio/apiary/pull/1234
```

blackbelt.handle_github.**merge**(*pr_url*)

This merges PR on Github into master:

1. Inspects the current repository and the pull request
2. Switches to master and brings it up to date
3. Merges the PR locally and pushes to master
4. Deletes the merged branch from the remote repository/github

TODO:

- Comment the associated Trello card

Deploying Pull Requests

blackbelt.commands.gh.**deploy_command**(*pr_url*)

Usage:

```
bb gh deploy https://github.com/apiaryio/apiary/pull/1234
```

blackbelt.handle_github.**deploy**(*pr_url*)

Deploys PR to production

1. Does *Merging Pull Request*
2. Inform people on Slack about the merge and the deployment intent
3. Prepares Heroku deploy slugs using `grunt create-slug`
4. Waits for CircleCI tests to pass
5. TODO: If they fail, asks for retry
6. Asks for deploy confirmation
7. Notify others on Slack about deploy
8. Deploys
9. Creates a release on GitHub, using merged branch name as 'ref'.
10. If it can figure out related Trello card (looks for "Pull request for <link>"), moves it to "Deployed by" column
11. Does *not* bring beer yet, unfortunately

Deploying master

Deploy current branch to production with:

```
bb production
```

This:

1. Informs others on Slack
2. Deploys master to production using `grunt deploy`
3. Deploys master to production using `grunt deploy --app=apiary-staging-qa`
4. Deploys master to production using `grunt deploy --app=apiary-staging-pre`

Rollback production

Rollback production to previous version with:

```
bb rollback
```

This:

1. Informs others on Slack
2. Rollback production using `grunt rollback`
3. Rollback staging-qa using `grunt rollback --app=apiary-staging-qa`
4. Rollback staging-pre using `grunt rollback --app=apiary-staging-pre`

4.1.3 Testing

Deploy current branch to staging with:

```
bb stage
```

This:

1. Discovers what the current branch is
2. Informs others on Slack
3. Deploys the branch to staging using `grunt deploy`

4.2 Story Owner

Story Owner is responsible for a specific user story that's going to be developed. Story lives as a card on a designated Trello Story Board.

He interacts with *Work Board* as well.

Those are the task *Story Owner* may want to perform.

4.2.1 Product list to Work Cards

Story Owner first breaks down the story cards into chunks by putting the list into the *Story*. After this is somehow done and run through with *Developer*, one usually wants to "transer" it to *Work Board* so it can be developed.

To help with this task, one can use this command:

`blackbelt.commands.t.`**`schedule_list`**`()`

Usage:

```
bb t schedule-list [--owner="TrelloUserName" [--story-list="Checklist Name"] [--
↪label="color"] http://trello.com/c/story-card-shortlink
```

Takes a TODO checklist on a given Story Card. Converts the items into the Work Overview Cards.

Story list defaults to "To Do", Owner (that the new work tasks are assigned to) defaults to you.

4.2.2 Clean sweep

blackbelt.commands.t.**migrate_label**()

Usage:

```
bb t migrate-label --label="Product: Example" --board="1KsoiV9e" --board-to=
↪"lEL8Ch52" --column-to="Prepared buffer"
```

4.2.3 Get ready for next week

blackbelt.commands.t.**next_week**()

Create new columns on the *Work Board*: *Deployed by* <*sunday*> and *Verified by* <*sunday*>:

```
bb t next-week
```

4.2.4 Verify a story

blackbelt.commands.t.**verify**()

Looks through a checklists on *Story*, see whether incomplete items refer to a card and whether the card is in *Deployed by* <*sunday*> and *Verified by* <*sunday*> column.

If so, ask to open them and then to verify them (meaning ticking the checkbox):

```
bb t verify http://trello.com/c/story
```

4.3 Glossary

Terms used in this documentation.

Story Owner Person responsible for a specific user story that's going to be developed.

Developer Person transmuting a wish into the code that's running in production.

Product Board Trello board with User Story cards.

Story

Story Card

User Story

User Story Card Single Trello card on *Product Board* that describes a single user story.

Work Board

Work Overview Trello board with deployeable tasks/units derived from User Story cards.

Work Card

Work Cards Trello cards on *Work Board* that describe a task/unit of work, derived from a user story.

4.4 Indices and tables

- genindex
- modindex
- search

Index