Ahoy Documentation

Release 1.1.0

Aashil Patel

September 07, 2016

Contents

Ahoy is command line tool that gives each of your projects their own CLI app with with zero code and dependencies.

Simply write your commands in a yaml file and ahoy gives you lots of features like:

- a command listing
- per-command help text
- command tab completion
- run commands from any subdirectory

Essentially, ahoy makes is easy to create aliases and templates for commands that are useful. It was specifically created to help with running interactive commands within docker containers, but it's just as useful for local commands, commands over ssh, or really anything that could be run from the command line in a single clean interface.

CHAPTER 1

Examples

Say you want to import a sql database running in docker-compose using another container called cli. The command could look like this:

```
docker exec -i $(docker-compose ps -q cli) bash -c 'mysql -u$DB_ENV_MYSQL_USER
-p$DB_ENV_MYSQL_PASSWORD -h$DB_PORT_3306_TCP_ADDR $DB_ENV_MYSQL_DATABASE' <
some-database.sql
```

With ahoy, you can turn this into

```
ahoy mysql-import < some-database.sql
```

More Examples

CHAPTER 2

FEATURES

- Non-invasive - Use your existing workflow! It can wrap commands and scripts you are already using.
- Consitent - Commands always run relative to the .ahoy.yml file, but can be called from any subfolder.
- Visual - See a list of all of your commands in one place, along with helpful descriptions.
- Flexible - Commands are specific to a single folder tree, so each repo/workspace can have its own commands
- Command Templates - Args can be dropped into your commands using `{{args}}`
- Fully interactive - your shells (like mysql) and prompts still work.
- Self-Documenting - Commands and help declared in .ahoy.yml show up as ahoy command help and bash completion of commands (see below)

CHAPTER 3

INSTALLATION

3.1 OSX

Using Homebrew:

```
brew tap devinci-code/tap
brew install ahoy
# For v2 which is still alpha (see below)
brew install ahoy --HEAD
```

3.2 Linux

Download and unzip the latest release and move the appropriate binary for your plaform into someplace in your $PATH and rename it `ahoy`

Example:

```
sudo wget -q https://github.com/devinci-code/ahoy/releases/download/1.1.0/ahoy-`uname -s`-amd64 -O /u
```

3.3 Bash / Zsh Completion

For Zsh, Just add this to your ~/.zshrc, and your completions will be relative to the directory you're in.

`complete -F "ahoy --generate-bash-completion" ahoy`

For Bash, you'll need to make sure you have bash-completion installed and setup. On OSX with homebrew it looks like this:

`brew install bash bash-completion`

Now make sure you follow the couple installation instructions in the "Caveats" section that homebrew returns. And make sure completion is working for git for instance before you continue (you may need to restart your shell)

Then, (for homebrew) you'll want to create a file at `/usr/local/etc/bash_completion.d/ahoy` with the following:

```
#! /bin/bash

: ${PROG:=$(basename ${BASH_SOURCE})}
```

```
_cli_bash_autocomplete() {
     local cur opts base
     COMPREPLY=()
     cur="${COMP_WORDS[COMP_CWORD]}"
     opts=$( ${COMP_WORDS[@]:0:$COMP_CWORD} --generate-bash-completion )
     COMPREPLY=( $(compgen -W "${opts}" -- ${cur}) )
     return 0
 }

 complete -F _cli_bash_autocomplete $PROG
```

restart your shell, and you should see ahoy autocomplete when typing `ahoy [TAB]`

CHAPTER 4

USAGE

Almost all the commands are actually specified in a .ahoy.yml file placed in your working tree somewhere. Commands that are added there show up as options in ahoy. Here is what it looks like when using the example.ahoy.yml file. To start with this file locally you can run `ahoy init`.

```
$ ahoy
NAME:
   ahoy - Send commands to docker-compose services

USAGE:
   ahoy [global options] command [command options] [arguments...]

VERSION:
   0.0.0

COMMANDS:
   vdown  Stop the vagrant box if one exists.
   vup    Start the vagrant box if one exists.
   start  Start the docker compose-containers.
   stop   Stop the docker-compose containers.
   restart  Restart the docker-compose containers.
   drush  Run drush commands in the cli service container.
   bash   Start a shell in the container (like ssh without actual ssh).
   sqlc   Connect to the default mysql database. Supports piping of data into the command.
   behat  Run the behat tests within the container.
   ps   List the running docker-compose containers.
   behat-init Use composer to install behat dependencies.
   init   Initialize a new .ahoy.yml config file in the current directory.
   help, h  Shows a list of commands or help for one command

GLOBAL OPTIONS:
   --help, -h     show help
   --generate-bash-completion
   --version, -v    print the version
```

CHAPTER 5

Version 2

All new features are being added to the v2 (master) branch of ahoy which is still in alpha and will have breaking changes with v1 ahoy files, so to use ahoy v2, you'll need to do the following:

- Upgrade to the ahoy v2 binary which currently needs to be compiled from source. If you are using homebrew, you can use that to upgrade to v2 using the following:

```
brew uninstall ahoy # Required or you'll get errors
brew upgrade # Updates the tap
brew install ahoy --HEAD # Installs ahoy by compiling the latest from the master branch
ahoy # You should see full version that you're using.
```

- Change your `ahoyapi: v1` lines to `ahoyapi: v2`

5.1 New Features in v2

- Implements a new feature to import mulitple config files using the "imports" field.
- Uses the "last in wins" rule to deal with duplicate commands amongst the config files.

```
commands:
  list:
      usage: List the commands from the imported config files.
      imports:
        - ./confirmation.ahoy.yml
        - ./docker.ahoy.yml
        - ./examples.ahoy.yml
```

5.2 Planned v2 features

- Provide "drivers" or "plugins" for bash, docker-compose, kubernetes (these systems still work now, this would just make it easier)
- Do specific arg replacement like {{arg1}} and enable specifying specific arguments and flags in the ahoy file itself to cut down on parsing arguments in scripts.
- Support for more built-in commands or a "verify" yaml option that would create a yes / no prompt for potentially destructive commands. (Are you sure you want to delete all your containers?)
- Pipe tab completion to another command (allows you to get tab completion)