m Documentation

Release release

Nathan Hoad

March 16, 2013

CONTENTS

m is a small command-line tool for making repository-based projects and tmux session management easier. It also helps to make feature branches and Mercurial's shared repositories easier to use.

It works by providing a succinct set of commands for creating or resuming work on projects, finding files within a project, and jumping to subdirectories quickly.

Put simply, I use Tmux and Mercurial a lot. In particular, I use shared repos and feature branches. This tool helps automate those particular tools, and can be considered a thin wrapper around them.

CHAPTER

ONE

TUTORIAL

1.1 What is m?

The `README` says it best:

m is a small command-line tool for making repository-based projects and tmux session management easier. It also helps to make feature branches and Mercurial's shared repositories easier to use.

It works by providing a succinct set of commands for creating or resuming work on projects, finding files within a project, and jumping to subdirectories quickly.

Put simply, I use Tmux and Mercurial a lot. In particular, I use shared repos and feature branches. This tool helps automate those particular tools, and can be considered a thin wrapper around them.

This is a short tutorial on using m. It covers off configuration, creating projects, working on them, and quickly navigating them.

1.2 Installation

Get a copy of the repository:

```
hg clone https://bitbucket.org/getoffmalawn/m
```

Make sure m is in your $PATH, or create an alias. I like to create an alias:

```
echo alias m='~/projects/m/m' >> ~/.bashrc
```

m also makes use of the yaml package for configuration, so you'll need that, too:

```
pip2 install --user yaml
```

Lastly, you'll need to make sure you have shared repositories enabled in mercurial. Add *share=* under `[extensions]` in your hgrc.

And that's that. Theoretically you're done, but you really need to configure m to actually use it. Onward!

1.3 Configuration

m reads a config file from your home directory, called `.mrc`. It's a yaml file. Yaml is nice.

Here's an example mrc:

```
projects:
    - ~/projects
    - ~/repos
commands:
    - vim
    - python2

presets:
    dotfiles*:
        commands:
            - vim
        replace: true
```

projects A list of the directories where you keep your projects. Please note that these are NOT the individual projects - that would be too much work!

commands A list of the commands to execute when creating a tmux session, one command per window, in the order listed.

presets Per project configuration. The names of the projects support globbing, so if you have multiple projects for the same codebase, prefix them with the same name to save a lot of copy-pasting. The presets can be configured to extend or replace the global configuration with the `replace` option.

1.4 Creating projects

m is built on the idea that you already have a bunch of projects you want to work on. As such, it doesn't facilitate creating *brand new* projects, only forking existing ones. This may be something implemented in the future as it becomes necessary.

You can create projects in two ways;

1. From a remote repository
2. From a local repository, using shared repositories

Either way, the interface is the same - the `new` command. There's an alias, `n`, to save you some typing.

Creating a project from an online repository:

```
m new https://bitbucket.org/getoffmalawn/m
```

Creating a project from a local repository:

```
m new m # create at path m-copy
```

To create a project and specify the destination path, simply suppliy another argument:

```
m new m some-great-feature
```

You can also specify the name of the branch to create, or the branch to start work in:

```
m new m branch_name=some-great-feature
m new m branch_from=1.5
```

Note: Creating a project from an online repository currently treats the target as an absolute path. This differs from using a local repository significantly - using a local repository creates projects in the same directory as the source.

This is considered a bug, but the solution is not clear. Feel free to offer solutions.

1.5 Resuming work on projects

Far more common a use case is resuming work on a project, thus it has the most cool stuff.

The basic work flow is this:

1. Change to the directory of a project
2. Start Tmux.
3. Open a bunch of common programs
4. Start working.

The `work` command aims to make this process easier, reducing steps 1-3 down to only one. As with the `new` command, there's an alias, `w`, to save you some typing. Using it is very similar to `new`:

```
$ m work m
```

The above command will do steps 1-3 of the outline above. Step 4 is something I can't automate. I'm sorry.

Note: You can actually omit the command entirely. The default command is `work` if no command is specified.

1.6 Finding files in your projects

Project navigation is the bane of software development. m tries to alleviate some of the tediom of this, providing two commands, `grep` and `vim`. They do what you think:

```
$ m grep foo # open all files containing "foo" in the file
$ m vim foo # open all files containing "foo" in the name
```

They also support multiple patterns, so you can do things like this:

```
$ m grep foo bar baz
$ m vim foo bar baz
```

The grep command, ultimately, wraps `grep`. So you can pass regular expressions along. The vim command, however, uses globs. Both commands won't open files that are under repository internal directories for Mercurial, Git, Bazaar or Subversion.

There's a limit on how many files m will open, though - if there's more than 10, it won't do it.

1.7 Jumping to subdirectories quickly

I like autojump, but I find it isn't smart enough - e.g. if I have multiple `tests` directories, it will often jump to the wrong one. m can help with this, too. In fact, this is what the seemingly useless `printdir` command is for!

So, imagine that you have directory in a dotfiles project, called iwilldiffer. You want to quickly jump to it to do some work that you can't easily do from where you currently are - here's what you type into your shell:

```
$ mcd dot iwilldiff
$ pwd
/home/nathan/projects/dotfiles/.vim/bundle/iwilldiffer
```

Isn't that nice? The way you configure it is a little gross... but here it is. Add this to your shell's config file (e.g. .bashrc or .zshrc):

```
mcd() {
    m printdir $1 $2 | read DIRECTORY
    if [[ "$DIRECTORY" != "." ]]; then
        cd "$DIRECTORY"
    else
      print "Couldn't find $2"
    fi
}
```

A little hacky, but it gets the job done.

1.8 Partial matching

m supports flexible partial matching on project discovery, to, again, save you some typing. For example, say you have a dotfiles project, you'd normally do something like this:

```
m work dotfiles
# or, if you want less typing...
m dotfiles
```

But you don't have to do that. As long as you specify the shortest unique name of the project, you can save yourself some typing:

```
m dot
```

If you also find you have projects that follow a Subversion like structure, like so:

```
$ ls ~/projects/some-project/
3.1 3.2 3.3 trunk
```

Where you may want to work on trunk, you could do this:

```
m some/trunk
# or even...
m so/tr
```

This allows you to jump to projects very easily. There's no limit to how many levels you can go with this.

Note: Partial matching is supported by a lot more than just `work`. You can use it with `grep`, `vim`, `printdir`, even `new` for the local source project!

CHAPTER

TWO

TODO

- Resolve `new` discrepancies for remote vs local repositories
- Create a stateful version of m, so that the project isn't necessary if you're already working on it.
- Tests.

CHAPTER

THREE

QUICKSTART

Here's a quick tutorial, for those who just want to try it out:

```
hg clone https://bitbucket.org/getoffmalawn/m
alias m=$PWD/m/m
cp m/mrc.example ~/.mrc
m dotfiles # start working on your dotfiles repo, which lives in ~/projects
```

Of course, m gives you much more than that. Check out the full *Tutorial* to see what else it gives you.

CHAPTER

FOUR

DEVELOPMENT

m is a simple project, with no real release schedule. It's essentially rolling release, until such a time that it needs to become more structured. If you feel that you can contribute something to m, please do so by submitting a pull request to the repository on bitbucket.