Sprinter Documentation

Release 1.1

Yusuke Tsutsumi

Oct 27, 2017

Contents

Sprinter is a tool to help create environment bootstrapping scripts and manage developer environments.

Here are some problems that sprinter was designed to solve:

- syncing up personal development environments across computers
  - syncing rc files
  - installing packages
  - configuring systems (e.g. git or ssh configs, setting up the PS1/shell prompt)
- distributing standard development tools and helpers across a company or organization
  - distributing common shell scripts
  - distributing third-party packages
  - distributing internal packages
  - performing strange on-time-setup quirks and workarounds when you can't get around to fixing it
- managing multiple development environments on a single machine
  - need to switch between personal and company-specific environment
  - need to switch between environments for open-source projects

Sprinter was designed with modularity, adaption, and cross-compability in mind. Some of the features of sprinter include:

- Installing environments directly from configs on the web
- Updating existing environments
- Managing several environments, activating and deactivating as needed
- Dynamically installing new functionality via formulas
- Sandboxing environments as necessary, such as brew or node.js

CHAPTER 1

Install Instructions and Tutorial

Please refer to the readme for instructions on installing sprinter.

It's a good first step to follow the *Sprinter Tutorial*

CHAPTER 2

Compatible Systems

Sprinter is currently actively developed against the following operating systems:

- OSX
- Ubuntu

And the following shells:

- bash
- zsh

However, Sprinter should work against Debian distributions, and most Ubuntu-based distributions.

Feel free to make a ticket with your difficulties with other unix-based operating systems.

There are currently no plans to develop sprinter against non-unix based operating systems (such as Windows). However, if you're feeling ambitious, post your thoughts in the Google Group.

CHAPTER 3

Questions?

Try our *FAQ*, or post a topic in the Google Group.

CHAPTER 4

Contents

Sprinter Tutorial

In this tutorial, you will learn:

- How to write a sprinter *manifest* file
- How to ask for user input (username, password, etc)
- How to use user input

Installation

First, you need to install sprinter. You can find install instructions in the sprinter readme here.

Build a sprinter configuration file

Each sprinter environment is completely defined by a sprinter configuration file. Think of this file as your main way of managing your sprinter environment: any changes you make here will be picked up next time you update your environment. Here's a good starting point for a sprinter config:

```
[config]
namespace = myenvironment

[git]
formula = sprinter.formula.package
apt-get = git-core
brew = git

[github]
formula = sprinter.formula.ssh
keyname = github.com
nopassphrase = true
```

```
type = rsa
host = github.com
user = git
hostname = github.com
```

What does this do? Well, give it a shot! Write this to a file called myenvironment.cfg (you should replace myenvironment with your username or whatever makes sense to describe your own personal environment), and install it with:

    sprinter install myenvironment.cfg

When you run the above command, you will first be prompted to configure sprinter if you haven't already.

The next thing sprinter will do is install the 'myenvironment' environment. As defined above, this consists of:

- install brew if you don't have it already (OSX Users)
- use brew or apt-get to install git
- create an ssh key just for github, and add it to the ssh configuration file

Now you just add the ssh key to github, and you're done! (you can find the path to the ssh file in your ~/.ssh/config file) (Unfortunately it's not possible to add the key to github programatically)

This outlines a lot of the basic functionality that sprinter provides:

- Multiple environments can be installed at the same time, with different specific names. In this case, we chose to name our environment 'myenvironment'
- adding environment configuration through 'features'. a feature is decribed by a section in the configuration file (besides 'config'). In this example, we have two features:
  - 'git', which installs git
  - 'github', which generates an ssh key

Now that's not super difficult, so let's try something more complicated.

sub is a command namespacing tool that allows the creation of subcommands. (e.g. moving to your workspace directory or running your server). Let's try adding this to our configuration.

Every feature needs a formula to define what the actual feature is going to do. sprinter.formula.ssh, as shown above, generates ssh keys. sprinter.formula.package installs packages from the appropriate package managers. So how about cloning git repositories? Luckily, sprinter has a formula for this as well: sprinter.formula.git. We can add a new feature by adding it's configuration into the environment config. We'll add a section to myenvironment.cfg now:

```
[config]
namespace = myenvironment

[git]
formula = sprinter.formula.package
apt-get = git-core
brew = git

[github]
formula = sprinter.formula.ssh
keyname = github.com
nopassphrase = true
type = rsa
host = github.com
user = git
hostname = github.com
```

```
[sub]
formula = sprinter.formulas.git
depends = github
url = git://github.com/mygithub/sub.git
branch = mybranch
rc = eval "$(%(sub:root_dir)/bin/sub init -)"
```

The git formula clones a git repository into sprinter's directory (typically ~/.sprinter). In the sub feature, we then evaluate sub's init script by injecting 'eval "$(%(sub:root_dir)/bin/sub init -)"' into one's .bashrc or .zshrc file.

You can get more information about each of the formulas, and what they do, on the *List of existing sprinter formulas* page.

Now remember at this point, sprinter already knows that you have an environment 'myenvironment' installed. Instead of running an install again, you can run an 'update' command on the environment:

```
sprinter update myenvironment
```

The environment 'myenvironment' knows where it found the file last time, and will record it's location for updating in the future. Although storing it locally is perfectly fine, it makes more sense to throw it online somewhere where all of your machines can access it. as an example, check out github user toumorokoshi's environment configuration file:

https://raw.github.com/toumorokoshi/yt.rc/master/toumorokoshi.cfg

variables in sprinter and referencing other formulas

In the above example, you'll see that you can reference variables and information about other formulas in the values set. In the 'sub' example, the value %(sub:root_dir)s in the 'rc' option gets replaced with the directory of the sub feature during execution. This can make it very easy to perform operations that rely on information about other features, or the global configuration.

Here's some examples of variables that are set in the above environment:

- %(sub:url)s resolves to git://github.com/mygithub/sub.git
- %(config:namespace)s resolves to 'myenvironment'

Grabbing user input

Sprinter also provides the capability to prompt the installer for input when installing a sprinter environment. Some common examples are:

- getting a username
- getting passwords for various services
- getting configuration options (version control root directories, workspaces)

You can grab user input by adding an 'inputs' option to any feature. Here's an example of getting a user's username, password, and git root then using it to make the git root and upload an ssh key through a rest api:

```
[config]
inputs = gitroot==~/git/

[create_git_root]
formula = sprinter.formula.command
install = mkdir -p %(config:gitroot)s
env = export GITROOT=%(config:gitroot)s
```

```
[stash]
inputs = username
         githostpassword?
formula = sprinter.formula.ssh
depends = curl
keyname = mygithost.com
nopassphrase = true
type = rsa
user = git
hostname = mygithost.com
install_command = curl -k -u '%(config:username)s:%(config:githostpassword)s' -X POST␣
↪-H "Accept: application/json" -H "Content-Type: application/json" https://mygithost.
↪com/rest/ssh/1.0/keys -d '{"text":"{{ssh}}"}'
use_global_ssh = False
```

Note the section 'inputs' has specific syntax:

```
gitroot==~/git/  # the == provides a default to the parameter ~/git/
username   # this is a standard, just asks for a username
githostpassword?  # the question mark makes it a hidden parameter on input, for␣
↪passwords and other sensitive data
```

If you run a sprinter install of this configuration, you would be prompted to enter the variables specified:

```
$ sprinter install sshexample.cfg
Checking and setting global parameters...
Installing environment sshexample...
please enter your gitroot (default ~/git/):
please enter your username:
please enter your githostpassword:
```

All prompted variables in the sprinter configuration are added to the config section, and can be used with %(config:MYVAR)s. In the example above, %(config:username)s will resolve to whatever the username parameter was.

When you update the environment in the future, you don't have to enter the parameters again. This is because sprinter environments remember parameters (except passwords/secret parameters. Sprinter stores values in plaintext, so it's never a good idea to store passwords that way.). If you want to re-enter parameters, you have to do an update with a –reconfigure:

```
$ sprinter update sshexample --reconfigure
```

rc and env

If you look at the configuration above, two parameters can be applied to almost all commands. Those are 'rc' and 'env'. rc and env handle the actual content that is injected into your shell (e.g. what goes in your .bashrc or .zshrc). For example, a GoLang installation requires some environment variables set. You can do so like this:

```
[golang-debian]
systems = debian
formula = sprinter.formula.unpack
executable = bin/go
symlink = go
remove_common_prefix = true
url = https://go.googlecode.com/files/go1.1.linux-amd64.tar.gz
type = tar.gz
```

```
env = export GOROOT=%(golang-debian:root_dir)s
rc = function go() {
        go version
    }
```

(the sprinter.formula.unpack formula handles unpacking of tar.gz, zip, (and dmg files for OSX)). Here we set an environment variables in 'env', and put functions in 'rc'. This ensures that environment variables are available for graphical applications, while function are available for shells.

It's ok not to get into specifics, most of the time just follow these rules:

- environment variables go into 'env'
- everything else goes into 'rc'

What next?

Congratulations! You know a majority of the functionality you need in sprinter. If you have questions about how to do specific things, try the FAQ or look at one of the doc pages, or post a question at our Google Group

Also check out the snippets section. This is a set of snippets that describe how to install common things like node.js

FAQ

Sprinter keeps overriding my custom *rc! How can I stop it?

Sprinter will always inject itself after everything in a profile or rc file, with the exception of text in a block surrounded by #SPRINTER_OVERRIDES. These will always run after any sprinter configuration.

How do I make a sprinter formula?

A sprinter formula is just a python module or egg that a python class extends the 'formulabase' class, located in sprinter.formulas.formulabase.

If you're not familiar with python, it's easier to just follow an example, like this one: https://github.com/toumorokoshi/yt.formula.node.

I need help! Who do I talk to?

If you have a question about a specific formula, it's best to pots a bug or talk to the author or the formula.

If you have questions about sprinter, your best bet is to post a message in the Google Group.

If there's behaviour that you think is a bug, you can also create a ticket.

Sprinter Examples

Here are some cool ways to use sprinter!

Sprinter patterns

A good pattern that developers tend to follow is to store all of their environment rc files (.emacs, .vimrc, etc) in a git repository, and clone and symlink the result. sprinter can automate that pattern. Look at this example section below:

```
[ytrc]
formula = sprinter.formula.git
depends = github,git
url = git://github.com/toumorokoshi/yt.rc.git
command =
    rm $HOME/.vimrc
    ln -s %(ytrc:root_dir)s/.vimrc $HOME/.vimrc
    rm $HOME/.screenrc
          ln -s %(ytrc:root_dir)s/.screenrc $HOME/.screenrc
    rm $HOME/.emacs.d
          ln -s %(ytrc:root_dir)s/emacs $HOME/.emacs.d
    rm $HOME/.viper
          ln -s %(ytrc:root_dir)s/.viper $HOME/.viper
    rm $HOME/.emacs
          ln -s %(ytrc:root_dir)s/emacs/.emacs $HOME/.emacs
    rm $HOME/.tmux.conf
          ln -s %(ytrc:root_dir)s/.tmux.conf $HOME/.tmux.conf
rc = . %(ytrc:root_dir)s/rc
```

Installing Sub

sub is a command namespacing tool that allows the creation of subcommands. (e.g. moving to your workspace directory or running your server). This works well with sprinter because:

- sub creates a clear, understandable namespace for shell commands
- sprinter downloads executable and dependencies, and updates the environment needed for those commands

Here's an example sub configuration section:

```
[sub]
formula = sprinter.formulas.git
depends = github
url = git://github.com/mygithub/sub.git
branch = mybranch
rc = eval "$(%(sub:root_dir)/bin/sub init -)"
```

List of existing sprinter formulas

The following are a list of existing sprinter formulas:

- sprinter.formula.command : a formula to run a shell command during a specific state
- sprinter.formula.eggscript : a formula to install a python executables that exists as eggs
- sprinter.formula.env : a formula to set environment variables
- sprinter.formula.git : a formula to clone git repositories
- sprinter.formula.package : a formula to install packages via native package managers
- sprinter.formula.perforce : a formula to setup perforce version control

- sprinter.formula.ssh : a formula to setup ssh keys and configuration
- sprinter.formula.template : a formula download a file, specialize it, and put the result where desired
- sprinter.formula.unpack : a formula to extract compressed files, and symlink desired executables
- yt.formula.node : install node.js and npm packages

Sprinter Formulas

Sprinter formulas are the building block of sprinter environments: they handle a specific piece of functionality of your environment, from cloning a git repository to install system packages.

In your sprinter environment configuration, each section (aside from config, which is intended for environment configuration) represents a configured formula, knows as a *feature*. Here is an example:

```
[sub]
formula = sprinter.formula.git
url = git://github.com/Toumorokoshi/sub.git
branch = yusuke
rc = temp=`pwd`; cd %(sub:root_dir)s/libexec && . sub-init2 && cd $tmp
```

This section:

- utilizes the standard sprinter git formula
- clones a git repo from the url specified
- checks out a specific branch yusuke
- adds the initialization command to the environment's .rc script

Configuration parameters vary from formula to formula, so look at the documentation to figure out which parameters are available to you.

Now let's talk more about formulas in detail.

Where to find formulas

Formulas are either included in the sprinter standard library, or can be sourced through python's pypi package repository, or through a url.

A list of available formulas can be found on the *List of existing sprinter formulas* page.

Standard Formula Options

Although sprinter formulas perform different functions, they all have a common set of functionality to facilitate workflows like adding files to the init script.

These functions are:

- 'rc': this will add lines into the .rc of your environment, thereby being added to your environment if it's activated. (for setup and updates)
- 'env': this will add lines into the .env of your environment, thereby being added to your environment if it's activated. (for setup and updates)
- 'command': this will run the specified command after the formula is finished (for setup and updates)

- 'systems': this specifies the systems that this particular formula should run on. The currently supported values are:
  - osx = OSX systems
  - debian = debian-based systems

Manifests

This covers the configuration required for a manifest, and the advanced aspects involved.

Special Sections

The following sections have a special meaning in a manifest:

config

The config section is where all user variables are stored. in addition, it can also house the name of the environment, input variables, and any standard confguration you need in your environment.

One can also add messages to the beginning and the end of sprinter with the following variables:

- message_success: print a message at the end of a sprinter command, on success
- message_failure: print a message at the end of a sprinter command, on failure

Variable substitution

Feature configurations in a manifest have the ability to reference each other: specifically, they utilized the following format:

```
%(FEATURE:PROPERTY)s
```

E.G. something like *%(git:configpath)s* will reference the 'configpath' property of the 'git' feature. In addition to other features, it is also possible to reference the config sections as well, with:

%(config:PROPERTY)s

Common usage examples for this include having a single username variable in the input, then resolving it into subsequent features with: *%(config:username)s*

Filters

Sprinter also supports some filters, to convert strings into a desired format. For example, to escape special characters, you can use:

%(config:password|escaped)

To reference the password variable, escaped. The escaped function uses the 're.escape' method in python.

Environment Lifecycle

The environment lifecycle was designed to be minimal and intuitive. Actions occur as they are necessary.

Installation

When an environment is installed for the first time, the 'install' directive is called on every feature.

Upgrade

When an environment is upgraded:

- newly listed features have the 'install' directive called on them.
- existing features have the 'update' directive called on them.
- features no longer listed in have the 'remove' directive called on them.
- features with a different formula will first have the old formula removed, and the new formula installed

Remove

When an environment is removed, every feature has the 'remove' directive called on them.

Deactivate

When an environment is deactivated, every feature has the 'deactivate' directive called on them.

Activate

When an environment is activated, every feature has the 'activate' directive called on them.

Sprinter Internals

This page discusses the internals of a sprinter environment. Specifically, the building blocks that constitute a sprinter environment.

Environment activation/deactivation

The main tool that environments are activated and deactivated is through "injections" of text into various configuration files on a client machine. A common injections that occurs is injecting the .rc file for an environment into a .bashrc/.bash_profile like so:

```
#sprinter-ENVIRONMENT
inject environment
#sprinter-ENVIRONMENT
```

Injections can be performed on any set of files that exist with an environment. An example of common ones are:

- the .ssh/config file
- .pypirc for local repositories
- .vimrc for vim configuration
- .emacs for emacs configuration

And more! Any file can have configuration injected, which should be removed in the activate/deactivate step.

The .rc file

Every sprinter environment has a .rc file at it's core. Identical in concept to a .bashrc or .bash_profile, this .rc file contains a majority of the configuration of the setup for an environment.

The .sprinter-ENVIRONMENT directory

A majority of the files required for a sprinter environment are stored in a .sprinter-ENVIRONMENT directory within

Features

Each section in a sprinter configuration represents a "feature", which exists in a particular state. Each service is dealt with separately, and is designed a service directory within the configuration root that it can use to place whatever it would like (clone a git repository, unpack a package, etc).

OSX Best Practices

- If your formula installs applications, move them applications into ~/Applications. This is to ensure that spotlight can find it, while containing sandboxing for a user.

Glossary

environment An environment in the sprinter context is a collection of *features* for a client machine that can be described by a sprinter manifest file. Sprinter's main job is to install, update, and ultimately manages these environments.

feature A sprinter feature represents a single unit of configuration for a sprinter environment. A feature should represent a single modular, functional unit to manage one aspect of an environment, such as the environment variables, a package that needs to be installed, or an in-house command line tool.

formula A formula represents a classification of a feature, that provides the steps to install, update, etc. a feature.

manifest A sprinter manifest is a configuration file describing a sprinter *environment*. Sprinter manifest examples can be found in the source code, or in the *Sprinter Tutorial*.

.rc file The file that is *injected* into the .bashrc or shell's rc for the client. This performs the majority of the activation and deactivation of a sprinter environment. More information can be found at *Sprinter Internals*.

.env file The file that is *injected* into the graphical environment or shell profile on the client. Similar to the .rc file, this is intended for configuration that specifically affects the client and not specific functionlity for a interactive shell (e.g. environment variables instead of shell functions) majority of the activation and deactivation of a sprinter environment. More information can be found at *Sprinter Internals*.

injection An injection is when sprinter-specific configuration is inserted into an existing configuration file on a client. More informmation can found at *Sprinter Internals*.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Index

Symbols

E

F

I

M