Tupelo User Guide

Stuart Maclean
Applied Physics Laboratory
University of Washington
mailto:stuart@apl.uw.edu
phone:206 543 1403

April 1, 2017

1 Introduction

This document describes how to use the Tupelo whole-disk imaging software that has been open-sourced and released on github (https://github.com/UW-APL-EIS/tupelo.git). The Tupelo interface is inspired by git, so experience with git is helpful.

2 Build

Though the build instructions for the software are in the main README, we'll review them here. Tupelo is a Java codebase and is built via the Maven build tool. A JDK 1.7 or higher is required to build the code, and Maven 3 is required also (at time of writing, author uses JDK1.7.0_111, Maven 3.3.9). Though some features of the code are platform-independent, there are places where the Java code relies on C libraries such as FUSE, which are Linux specific. So this User Guide is really geared towards Linux/Unix users.

To grab the Tupelo code, clone from github:

```
$ git clone https://github.com/UW-APL-EIS/tupelo.git
```

As of March 2017, the develop branch is current, so switch to it and invoke Maven to build. The code is organized as several Maven modules:

```
$ cd tupelo
$ git checkout develop
$ mvn install
```

3 Command Line

Tupelo currently has a single entry point, the command line. Tupelo GUIs are feasible, but outside the scope of this guide. Tupelo is accessed like git. There is a single command-line entry point, called `tup`, and many subcommands. The `tup` program is simply a shell wrapper around a Java invocation. Verify that it runs on your system. If it fails, perhaps you don't have the Java system on your path. Invoking tup should list out all the possible sub-commands.

```
$ cd /path/to/tupelo
$ cd cli
$ ./tup
Usage: tup [-c configFile] [-v] <command> [<args>]

Commands
  help            Explain each command
  config          Show configured devices and stores
  device          List, add or remove device associations
...
```

4 Objects: Devices and Stores

Tupelo is a 'whole disk management system'. This just means that the unit of currency in Tupelo is a whole disk, rather than a file or filesystem. We talk of 'imaging a disk', 'capturing a disk' or 'storing a disk'. These are all equivalent — we make a logical copy of a disk of interest and store that copy in some non-volatile, read-only place. We call that place a Tupelo store.

4.1 Stores

Central to Tupelo is the notion of a 'store', which is just a database where disk contents are stored! Our first store implementation is called a 'file system store' and uses, as you can probably guess, a directory structure under a root directory to hold the disks. When we say database, we do *not* mean MySQL or any form of RDBMS, we just mean a mechanism for storing and later accessing chunks of disk data.

Currently Tupelo can access a local directory store and a store over http(s). Stores are given urls, very much how git uses urls for its remotes. More on this later.

4.2 Devices

To image disks with Tupelo, you 'push' the disk to a store, much like how it git you push your local repo to some upstream remote. Officially, in Tupelo we speak of devices rather than disks, but the two terms are equivalent. A device, then, is a complete hard drive you wish to capture. Three devices types are supported:

Physical Disks A whole hard drive, such as C: or /dev/sda. Tupelo interrogates the disk at a low level (using e.g. SCSI/ATA commands) to extract from the disk its vendor, serial number. etc. An example is 'ATA/SAMSUNG HD253GJ/S26CJ90B203913'. It is hoped that this disk and *only* this disk would ever exhibit that identifying string.

Disk Images A file on disk that represents some whole or part of a disk. For these disk type, we just use the file name for our diskID, conceding that this is hardly globally unique (we could have many image.dd files!). Disk Images are really only intended for testing/experimentation in Tupelo.

Virtual Disks The whole disk contents of a virtual machine under a VM Manager such as VirtualBox or VMWare . A VM's 'hard drive' is actually just a file (or set of files) in the host system. The .vdi or .vmdk files in your VirtualBox vm workspace are virtual disks. Tupelo has code that can 'see inside' these .vdi/.vmdk files to get at the contents of the hard drive as seen by the VM (Note: not fully done yet!)

Next, we'll review our Tupelo 'configuration'. The configuration is just a set of local names you can give to stores and devices.

5 Imaging Virtual Machine Disks

Much of the development and testing of Tupelo was done against virtual machine disks. Tupelo understands VirtualBox and some types of VMWare virtual disk formats (via the vmvols Maven project, see github.com/UW-APL-EIS/vmvols-java.git). So if you have some virtual machines to use as data, this is a great way to test Tupelo out. For purposes of this guide, we'll assume I have a VirtualBox VM called Win7_64-base:

```
$ cd /home/stuart/VirtualBox\ VMs/Win7_64-base
$ ls
Logs  Snapshots  Win7_64-base.vbox  Win7_64-base.vbox-prev  Win7_64-base.vdi
```

As is implied, this is a Windows 7 VM, with a single disk which has two NTFS partitions. We'll ingest this system into Tupelo.