**UNIX Proposal**

**Automated Environment Backup**

**Abstract** There will be a day when a hardware replacement is necessary, which means that your own personalize environments will be restored to default. As developers might install tons of plugins in order to suit their working environment, it’s always important to backup our configuration files to the cloud services. This proposal plans to provide a tool that will automatically backup all installed plugins from nodeJS, python, ruby, vim, sublime, and atom. Besides, backup files can be restored with just a single click; plugins will be installed automatically. We will first check for all installed plugins that is usually located within a list. We will then provide a Linux-based GUI front-end implemented by GTK+ or Qt for easy usage. Additionally, we can also use the personalize environment from third party provider, with option to choose which plugins to install. For more details, please refer to the following sections.

**PROJECT GOALS & MOTIVATIONS**

This project intends to implement a back-end backup process that could periodically backup configuration files in the background, as well as a GUI front-end that simplify the operation of backup and choosing which version to restore. The back-end backup process is able to backup all plugins installed on a system. Besides, the backup file can be shared on websites such as GitHub for others to use your personalized environment. For instance, if a system have vim and sublime installed along with tons of plugins, we’re able to restore all of it with just a single click; as well as backup file from third party provider. Additionally, if you ever wonder what plugins does a professional Network administrator installed on their system, just search for their shared backup file online, and try it on your own system. We will also provide a front-end friendly user interface for users.

We will first let users to set the backup time. Then, the backup will begin periodically. The expected output file is a text file which is up to 100MB large. This text file will then be stored in database. The language is assumed to be in English. The output file will content the name of all plugins installed on that particular system. Installed plugins can usually be found in a list located in the correspond application. For instance, plugins installed on Ruby is contained in a list and can be viewed using the command "gem list". We will then get the required information from that list, including the version of the plugins. During the restoration process, we will install every plugins listed on the output file, and automatically install it. If plugins update are available, then the backup version of the plugins will be installed instead. Users will have the option to choose which plugins to install.

For the time being, we have been uploading our own configuration files to GitHub to backup our installed plugins. This is not a very good idea as sometimes we might accidentally loaded into conflict plugin if we are careless. It will be very difficult for us to figure out which version to restore. It’s absolutely frustrating. This is the main reason why we’re doing this proposal. We hope to have an automated environment backup system that could manage plugins and develop environment with ease. We are really excited about this project since it will help us to solve one of the most headache problem in real life. It will absolutely come in handy when you need it. We are also planning to make this project an open-source project for public. The source code and prebuilt binary packages will be announced and released on the well-known SourceForge (sourceforge.net/) or GitHub (https://github.com/).

This project is challenging to us since it involves many criteria to consider about. For instance, not every plugins can be installed with just a simple command, different plugins might require different operations, such as, the installation of YouCompleteMe (a.k.a. YCM) in vim require compiling and setting up preferences. Besides, we also need to know what software is installed and their correspond plugins. While we have a theory solution to the problems that we have discovered, it will be interesting for us to try it out in real life. Additionally, we are also interested about the usage of GitHub API. We hope to have a better understanding of Linux as we implement this project.

**PROJECT ORGANIZATION**

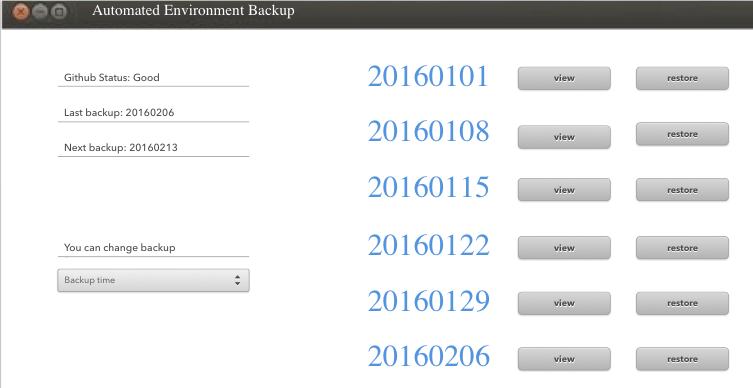


Fig.1 A sample GUI

The front-end GUI uses Qt or GTK+ to show the current backup information, current state of GitHub, GitHub API, and also an option for user to choose backup time on the left side. While on the right side, it will contain information about the latest 30 backups along with two buttons, restore and view. We will double confirm with the user if restore button is pressed to avoid any miss-click. The view button is to show detailed information of a particular backup.

When the restore button is pressed, we will first compare and show the differences between the current version and the version to-be-restored. There will be 4 different cases:

1) plugin exist and both version are the same.

* a) do nothing

2) plugin exist and both version aren’t same.

* a) double confirm with user about the differences between version
  + i) update plugin: reinstall plugin
  + ii) use the original version: do nothing

3) plugin doesn’t exist

* a) install plugin

4) couldn’t find the version of a plugin

* a) install the latest version

We will display a list to show what plugins will be installed. Before the restoration process end, we will do a backup.

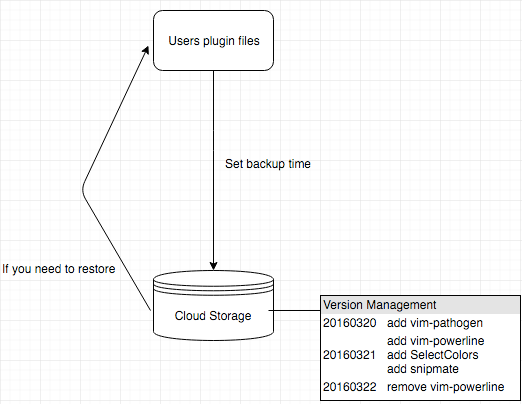


Fig.2 Project organization

The back-end program use crontab to implement the backup periodically at a given time stamp. User will also be prompted to log in into GitHub to backup on the cloud server. Back-end program will search for plugins of a particular software that is usually located in a list of that software. For instance, ruby uses the command "gem list" to show all installed plugins. However, not every software uses the same method to get the installed plugins. Hence, we will get the required information of installed plugins with different methods according to different software, as listed below.

* Ruby: Using the command "gem list"
* Python: Using the command "pip list" through pip
* Node JS: Using the command "npm ls"
* Vim: Use Vundle in .vimrc to manage all plugins
* Atom: Stored in directory "~/.atom/packages"
* Sublime: Stored in directory "/Users/{user}/Library/Application Support/Sublime Text 2/Packages"
* XCode: Stored in directory "~/Library/Application Support/Developer/Shared/Xcode/Plug-ins"

We assume that the information of installed plugins are stored in default location. Furthermore, we are planning to support more application in the future.

The difficulties arise when there are tons of plugins with different version installed. We need to ensure that we will get the correct version of the plugins. Besides, the installation of different software, such as, compiler and different programming languages are significantly different from one another. Therefore, in our crawling design, our goal is to ensure the correctness of data we get.

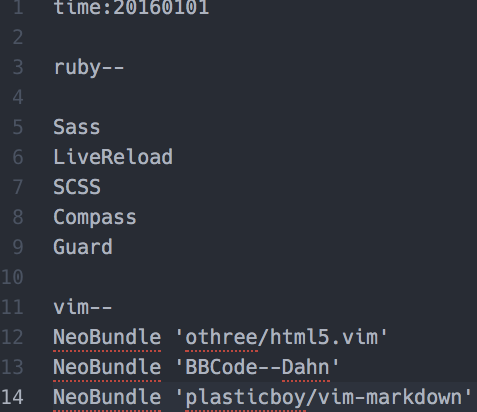


Fig.3 Output backup file

After data crawling process is done, we will create a document with a given time stamp, as shown in Fig.3. Then, we’ll output a text file, which will automatically upload to the GitHub. In order to ensure the consistency of data, we are using the following naming method:

* -env\_name
* plugin\_name plugin\_ver

We will periodically check for this data through the command "git add", and also "git commit" to ensure that data is stored in git repository. If the version of the plugins appeared to be in conflict, then we will compare the version of the plugins using the command "git diff". For instance, respiratory of version 1 contained a, b, c, and d, while version 2 only contained a, b, and c, git diff will show that d wasn’t installed in version 2. Besides, we will only update the backup file if there is any changes to reduce any unnecessary burden to Linux.

**TOOLS**

* The development environment: Ubuntu 14.04 LTS.
* The deployment environment: Ubuntu 14.04 LTS.
* The programming languages and compilers/interpreters used: C++/python/g++.
* Other:
  + Vundle: use to manage plugin in Vim, and also provides a checklist so that we backup easily.
  + GitHub: a fast, flexible, and collaborative development process that let you work on your own or with others.
  + pip: PyPA recommended tool for installing Python packages.
  + npm: a package manager. Installs, publishes and manages node programs.

**COLLABORATION**

* 1013353 吳冠賢：25% in total
  + Team leader
  + Write proposal
  + Develop the front-end program
    - Develop User Interface (UI)
    - Compare the version of plugins
  + Make the slides and poster for demonstration
* 1023350 馬少驄：25% in total
  + Develop the back-end program
    - Restoration process
    - Debugging
  + Prepare for the oral presentation
* 1023352 羅治杰：25% in total
  + Develop the back-end program
    - Backup procedure
    - Check which plugins are installed
  + Write and review proposal
* 1023353 蕭靖輝：25% in total
  + Develop the front-end program
    - Github API
    - Develop UI
  + Write and review proposal

**PROJECT SCHEDULE**

* Feb. 26–Mar. 12: Team up with the members.
* Mar. 13–Mar. 26: Write proposal.
* Mar. 27–Apr. 23: Implement front-end version control log, back-end backup mechanism
* Apr. 24–May 7: Combine the front-end to the back-end
* May 8–May 21: Debug the entire project.
* May 22–Jun. 11: Make a presentation and a poster for demonstration purposes.
* May 23: Prepare for presentation.