Software Management System

Table of Contents

Introduction 3

Installation 4

Directory Structure 5

Development Standards and Conventions 6

Implementation 7

swaddbuild/swaddpackage 7

swaddbuild.py 7

swaddpackage.py 7

addbuild\_addpackage.py 7

swlog 7

swlog.py 7

log.py 7

swbuild/swlink/swtest 7

swbuild.py 7

swlink.py 7

swtest.py 7

rebuild\_relink\_retest.py 7

swduplicate 7

swduplicate.py 7

duplicate.py 7

swreport 8

swreport.py 8

report.py 8

reportconform.py 8

reporthtml.py 8

reporttext.py 8

swversion 8

swversion.py 8

version.py 8

implementation and helper modules 8

swadm.py 8

config.py 8

Web Integration 9

Drupal 9

# Introduction

The NCCS Software Managements System is a set of scripts written in both Python and shell that are designed to ease and automate the task of installing, maintaining, and documenting software installations on HPC systems (or any system where large amounts of software are compiled from source). The NCCS Software Management System (hereafter referred to as swtools) allows many actions that would traditionally be incredibly labor intensive to be completely automated. For example:

* Rebuilding on applications on a system
* Automating rebuilds after a compiler upgrade
* Automating rebuilds after OS upgrades
* Dynamically generated HTML Status Pages
* More…

If there are any questions of any kind feel free to send me an email and I will certainly do my best to assist you.

Nick Jones

njones11@eecs.utk.edu

# Installation

As of the time of this writing, all copies of swtools are currently stored on the NCCS subversion server. To check out the current working copy of swtools, you will need to have a basic understanding of subversion and you will also need to have been granted access to the SVN server. Once this is complete, type:

svn checkout <https://svn.ccs.ornl.gov/svn/scicomp/sw-tools>

When prompted, type in your passcode. For more information on the NCCS SVN setup, see:

<https://wwwadm.ccs.ornl.gov/doku/svn/svn_index>

Once you have checked out the current working copy of the SVN, move the files to the location you would like them to reside. In order to run swtools, you will need to have at least Python 2.5 installed on your system. This software was developed using Python 2.5.1, and although no major incompatibilities are expected with future Python versions (except possibly Python 3.0), problems may occur at a future date. If this occurs, I advise you to check the online documentation for the version of python you are using for any warnings or suggestions regarding incompatibility with previous versions.

Assuming that you have the proper version of python installed, and that you have swtools installed correctly, you should be able to run any of the swtools scripts immediately. If this is not the case, check to make sure that you have the proper python module loaded. Although most NCCS systems have a version of Python greater than or equal to 2.5, almost all of the systems require that you explicitly load the python module for the version you desire.

Once python is loaded and you have downloaded swtools, the next step is to modify <install location>/bin/sw\_config to match the parameters of your local environment. I will not go into depth explaining the variables in sw\_config here. Each line is extensively commented, and should be all the information you need. Once this is complete, you are ready to use the swtools.

# Directory Structure

swtools has five main directory levels: root, architectures, applications, versions, and builds. The root level is the top level folder which contains all other swtools folders and applications. On Jaguar (at the time of this writing), the root level is /sw. Inside /sw, there are folders for each “architecture”. We define “architecture” to be any group of applications that are used by a machine or by a group of similar machines. Thus, at the NCCS, we have a single architecture called “xt” which all of our Cray XT3, XT4, and (hopefully) XT5 systems use. At the same time, we have architectures for individual machines such as “ewok” and “lens”. Inside each architecture folder, there is a folder for each application installed on the machine. Inside each application folder, we have a folder for each version of the software. Inside each version folder, there is a folder for each build of that version.

An example directory structure, NCCS JaguarCNL, April 1, 2008:

nai@jaguar1:/sw> ls

analysis-x86 bgp ewok smoky sources tools tools\_dev xt

nai@jaguar1:/sw/xt> ls

arpack atlas aztec bin blacs blas cmake craypat doxygen ferret fftpack fftw fpmpi gamess gcc globalarrays gnuplot grace hdf5 hypre idl imagemagick java-jdk java-jre lammps lapack metis modulefiles mumps namd ncl nco ncview netcdf p-netcdf papi parmetis pathscale

petsc pgi pgplot pspline python scalapack sprng subversion sundials superlu

nai@jaguar1:/sw/xt/hdf5> ls

1.6.5 1.6.6 1.6.7 1.8.0 description support versions

nai@jaguar1:/sw/xt/hdf5/1.6.7> ls

cnl2.0\_gnu4.2.1 cnl2.0\_gnu4.2.1\_par cnl2.0\_pathscale3.0 cnl2.0\_pathscale3.0\_par cnl2.0\_pgi7.0.7 cnl2.0\_pgi7.0.7\_par

hdf5-1.6.7.tar

nai@jaguar1:/sw/xt/hdf5/1.6.7/cnl2.0\_gnu4.2.1> ls

bin dependencies doc hdf5-1.6.7 hdf5.pbs hdf5.pbs.o91233 hdf5.pbs.o93419 hdf5.pbs.o93813 include lib rebuild relink retest status test.log

See the CUG Paper and the Wiki for more details on this.

# Development Standards and Conventions

Random Notes:

* All paths that are returned by a function do not have a trailing slash.
* Implementation and command line access have been separated in all scripts.

Implementation Details:

All of the scripts have had their command line access separated from the implementation.

swaddbuild.py >> addbuild\_addpackage.py

swaddpackage.py >> addbuild\_addpackage.py

And so on.

Almost all scripts use swadm.py and config.py . swadm.py is where all helper functions are stored. This contains functions like get\_versions() and error(). config.py contains functions that read fields from sw\_config. The only functions that are in config.py are ones used to read from sw\_config.

# Implementation

## swaddbuild/swaddpackage

### swaddbuild.py

Driver script for swaddbuild

### swaddpackage.py

Driver script for swaddpackage

### addbuild\_addpackage.py

Contains the implementation of addbuild and addpackage. This script is very readable and well commented. Should have no problems updating/modifying it.

## swlog

### swlog.py

Driver script for swlog

### log.py

Relatively simple script that just asks questions and then mails the answers to RT. The only thing that might possibly cause trouble is the smtplib stuff used to send the email. However, I refer you to Python’s doc if that happens.

## swbuild/swlink/swtest

### swbuild.py

Driver script for swbuild

### swlink.py

Driver script for swlink

### swtest.py

Driver script for swtest

### rebuild\_relink\_retest.py

Contains the implementation of rebuild/relink/retest. As these scripts go, it is relatively readable. It executes in a relatively linear fashion, and hopefully will be easily maintainable.

## swduplicate

### swduplicate.py

Driver script for swduplicate

### duplicate.py

Contains the implementation of swduplicate. Nothing too difficult here.

## swreport

### swreport.py

Driver script for report.py

### report.py

Calls reportconform.py, reporttext.py, or reporthtml.py based upon the action specified to it by report.main

### reportconform.py

The main control structure is a set of multiply nested loops. Basically, it loops through a listing of all of the files at the app level, then the version level, etc. During the course of each loop lots of different checks are performed on the files. There is some dangling text at the bottom to be used for sending NAG emails to installers. It never got fully implemented, which I apologize for. I would finish or disable it, but I don’t want to risk breaking anything at this point. Hopefully someone will come behind and finish it.

Conform uses the apperr/vererr/builderr flags to detect if a problem has been detected with the current app/version/build. If the flags are false, then this app will not get printed out during the report. If the flags are true, then they will be printed.

Also, take note of the recurse option. This uses python’s os.path.walk routine to recurse into the bottom level of all directories and check permissions.

### reporthtml.py

This is a hairy script – I’m sorry for that.

The script begins by looping through all the architectures. As it does this, the first thing that the script generates is the alphabetical table that lists all machines. However, in the middle of this loop, it generates the machine specific pages (both the alphabetical and category views). Once the big table with all of the machines has been generated, it generates the big table in a category view.

Main Loop (Table for all machine)

Generate Machine Specific Pages – Both Alphabetical and Category

Generate Application Pages

End

Second Loop

Generate Category Page for all machines

End

### To modify this script, you will need a decent knowledge of both HTML and Python.

### reporttext.py

This script is not too bad, mainly just a set of nested loops combined with a little bit of file i/o. Nothing too complicated here.

## swversion

### swversion.py

This is the driver script for version.py

### version.py

The only part of version that may seem complicated are the email functions and the date comparisons. If you are modifying this script, I advise you to become familiar with Python’s datetime module and associated datatypes. The email stuff is relatively simple – just Google Python’s smtplib module for examples.

## implementation and helper modules

### swadm.py

swadm.py implements all of the helper functions used throughout the swtools. All functions should be in this module, except the config file related functions. A few functions of note:

error – This is my error message function. It makes all error messages have a standardized output format.

get\_versions – Pass it a list of application paths and the version tag you want, and it will give you back a list of versions.

### config.py

config.py implements all of the funtions that read from sw\_config. All of the functions in this file are used to extract data from sw\_config in one way or another.

# Web Integration

‘swreport html’ currently outputs its generated html files to /sw/tools/www. This is configurable in sw\_config. To get these pages to appear on the website, we have to write a custom wordpress template. All of the templates that have been written to date are in /sw/tools/php . If you only want to display the software list for a single machine, make a copy of softwarestatusBGP.php and then modify it. Essentially, you need to replace all the instances of “bgp” with the name of the appropriate swtools architecture. Additionally, you need to modify the wordpress template name at the very top of the page. This is the name that you will see when you go looking for this template in the wordpress web interface.

If you only wish to display the large table that shows all software for all machines, then use softwarestatusall.php .

Once you have created the proper template, you now need to install it. All templates for the production webserver reside in:

/ccs/doc/wwwnccsgov/wp-content/themes/nightfall

This is where you need to place your template. Once this is complete, login to wordpress and create a new page. For the page template, select the template that you just created. It should be an option in the drop down list. The wordpress software status pages may not render correctly until published. This has to do with the fact that ‘swreport html’ outputs relative links. Relative links are not always compatible with the preview URL generated by wordpress.

<https://www1.nccs.gov/?page_id=868> is a typical link to a non published wordpress page. A relative link like “?&view=category” will generate <https://www1.nccs.gov/?&view=category> which is not what we want.

However, once the page is published, it will have pretty URL like <http://www-lb.nccs.gov/computing-resources/lens/software/> . If you use relative links here, they will work correctly and generate proper output like <http://www-lb.nccs.gov/computing-resources/lens/software/?&view=category> .

## Drupal

Drupal should be relatively easily integrated with the existing infrastructure. To create a Drupal software page, log in and make a new page. At the bottom, select PHP as the language you wish to write the page in. Once this is done, you should be able to write direct PHP code into Drupal.

Something like this will generate the proper page for the “all machines view”. Similar pages could easily be created by simply taking out the relevant block of code from the wordpress template.

<?php

if (($\_GET["software"] == "") && ($\_GET["view"] == "") ){

include('/sw/tools/www/alphabetical.html');

}elseif ( ($\_GET["software"] == "") && ($\_GET["view"] == "category") ){

include('/sw/tools/www/category.html');

}elseif ($\_GET["software"] != "" ){

include('/sw/tools/www/' .$\_GET["arch"]. '/' . $\_GET["software"] .'.html');

}

?>