ISGTC: an alternative to ~bofh/bin

David Schweikert <dws@ee.ethz.ch>

presented at the SANE 2004 conference on the 2004-10-01

1 Introduction

Scripts let us do system administration in an efficient and reliable way. These scripts are so useful to us because they are powerful. But like `rm -rf`, which is undeniably very useful, they are also very dangerous because of that power. One mistake and a disaster might happen.

That's why it is important for this kind of tools to be dependable. Not only do they need to work reliably, they also must have an understandable and consistent user interface to minimize the risk of human mistakes. This can be achieved by having a standard on how administration scripts are developed and how they should behave.

At the ISG.EE, a system administration group at the ETH Zurich, we never found the time and courage to write such a standard and decide that all our administration scripts should follow it. There were many choices to be made and having to select what is best for us among many possible variants, blocked us from achieving such a standardization.

Finally, in the year 2001, we came up with the *IT Support Group Tool Chest* (ISGTC). The ISGTC contains 37 *rules* that must be followed when writing a new tool and contains more than 100 administration scripts that follow those rules. All the scripts are released under the GNU General Public License (GPL). I will describe some of them in section 4.

2 Guiding principles

As system administrators, we take a different approach to computers than other users. We try to understand why computers behave the way they do, write scripts to automate repetitive tasks, do optimizations so that the services are as reliable and useful to the users as possible, ... While working as system administrator, we gather experience on how administration should be done.

At the ISG.EE we did formulate some basic principles of system administration, that are based on our experience. This section is a short summary of these principles, which form also the basis for the administration script policy that I will describe later on.

2.1 Documentation

Documentation is one of the most often neglected tasks of a system administrator. It is very challenging to keep up with user needs and at the same time to document the current state of the managed system.

Documentation is however also, in my opinion, the most basic requirement for reliable and efficient system administration. Only with proper documentation I can make sure somebody else can understand how to run the system. Even I will eventually forget all the details about something that I setup today.

2.2 Reproducibility

We control the systems we manage, and not the other way around. They work the way they do as a result of controlled administration and not of as a result of "random" interactive changes. We can re-install from scratch any of our machines exactly as they were before.

Besides greater reliability, reproducible system administration also means that we work more efficiently, since we can rapidly repeat tasks in case of hardware upgrades or if we want to do a similar setup somewhere else.

Poor reproducibility is the main main drawback of interactive configuration tools. If I click-enable an entry in some context sensitive menu, I might forget to do the same the next time around.

If a system is properly documented, its setup is also reproducible. The documentation just includes the necessary human steps for replaying what was done before.

2.3 Versioning and Accountability

In order to guarantee that useful information isn't lost, we keep the full history of the documentation and configuration files through a version control system such as Subversion. We can retrieve an old version if for example a mistake was done and we can look at the history of the system documentation to see what was changed to the system. When many people work together it allows them to keep up with each other.

Version control also allows all the administrators in the group to work in parallel on the same set of files and we can track who did what change to what file. This is very useful to aid in debugging when something doesn't work anymore or is not clear to somebody: you can find out who did the change and ask him/her directly.

2.4 Automation

Doing the same thing again and again is not only inefficient, but also boring and thus error-prone. If there is a pattern of actions, I can automate them with a script. This will make the required work faster and more reliable since those repetitive tasks will always be done the same way.

If some procedure is automated, I also need less to document, since I only have to describe how to start the script instead of every single step.

2.5 Redundancy avoidance

One of the worst enemies of a system administrator is redundancy. As soon as the same configuration information lives in two files, there is a potential problem of inconsistency. We might make a change in one file and forget to update the other. Therefore we try to avoid redundancy whenever possible. If the same information needs to be in different places, then we make sure that it is clear which one is the authoritative source and possibly automatically generate the others.

3 The ISGTC policy

I will now explain the most important rules in our administration scripts policy. The full text of the policy is also available on the ISGTC home-page (see the address at end of this article).

3.1 The isgtc tree and relocatability

By default the ISGTC scripts and configuration files are stored in`/usr/isgtc` with the following sub-directories:

`bin`	tool binaries (*bindir*)
`lib`	libraries (*libdir*)
`etc`	configuration files (*sysconfdir*)
`share`	read-only data (*datadir*)
`man`	man-pages (*mandir*)
`var`	read/write data (*localstatedir*)
`src`	sources for the tools and libraries (*srcdir*)

While the ISGTC has a standard installation location, every script is *relocatable*. ISGTC paths such as `/usr/isgtc/etc` are not written explicitly into the scripts. Instead special *magic* variables such as `ISGTC_MAGIC_BINDIR` are used. The

ISGTC could, for example, be configured to have binaries in `/opt/isgtc` and configuration files in `/etc/opt/isgtc`.

3.2 Sources and version control

The *sources* of the scripts are located in *srcdir*. Every script or library has a directory, which contains the source code and related files. We call these directories *modules*.

With the exception of *sysconfdir*, which contains the configuration, and *localstatedir*, which contains data saved by the scripts, the other ISGTC directories are populated exclusively by installing modules from *srcdir* using the installation script `isgtc_install`.

Because the files in other directories are derived from it, *srcdir* is the only directory under version control.

3.3 Ownership

Every module has a defined *owner*, who is responsible for maintaining it.

There is also an *ISGTC owner*, who is responsible for keeping the ISGTC as a whole in good shape. He/she ensures, that the ISGTC doesn't contain scripts with similar functionality or obsolete tools and also makes sure that the policy is respected.

3.4 META file and installation of modules

`isgtc_install` reads a description of the modules and what files to install from a file called `META` inside the module directory in *srcdir*. It configures and installs binaries, libraries, man-pages, and data files.

Listing 1 shows as an example the `META` file of the `diskadm` module.

The `*** install ***` section specifies that `src/diskadm/diskadm` should be installed as `/usr/isgtc/bin/diskadm`. In addition, the man-pages written in POD, the Perl inline documentation format, will be extracted and put in `/usr/isgtc/man`.

The `META` file also contains information about the owner of the module, what version of the policy document was used, on what platform it runs (either *unix* or *win32*), what kind of module it is (*tool* or *library*), and what other modules it depends on.

You can think of source modules as sort of packages like RPM, where the `META` file is the description of the package.

3.5 User documentation

Every module must have user documentation in POD format, which can be converted to man-pages and a number of other formats like HTML. If possible, the

Listing 1: /usr/isgtc/src/diskadm/META

```
policy      = 1.2
name        = diskadm
type        = tool
platform    = unix
description = Disk administration tool
pod         = diskadm, diskadm-design.pod
owner       = dws

*** install ***

# perm/owner    source          dest
755 root:root   diskadm         ISGTC/bin

*** depends ***

ISG_Util
ISG_DisksConfig
ISG_Solaris_DiskUtils
```

documentation is kept in the script itself so that it is easier to keep the documentation and code synchronized. The `isgtc_install` tool will verify that documentation is available and that it contains mandatory sections, such as `SYNOPSIS` and `COPYRIGHT`.

3.6 Options

The GNU syntax for long-options is supported by every tool and the following options are mandatory:

`--help, -h`	Help on usage of the tool.
`--version`	Version information.
`--noaction, -n`	Do not do any changes to the system, just show what would be done. If not implemented, the script must die with an error message.
`--verbose, -v`	Be more verbose about what is done. This option may be ignored.

Consistent options are important to ensure predictable behavior of scripts. Imagine the account removal script doesn't follow this rule and uses `-n` to activate the *no-questions-asked* mode.

3.7 Site independence

All scripts in the ISGTC must be *site independent*. This means that they must not contain installation-specific information such as hostnames or paths. In that way, we can make sure that configuration information really is in *sysconfdir* and that we can reuse the tool in other setups. Site-independence also means that **you** can use them too.

3.8 Configuration files syntax

Configuration files are placed in *sysconfdir* (by default `/usr/isgtc/etc`) and are called either just `toolname.conf` or alternatively `toolname/xxy.conf` if more than one file is necessary.

It should be possible to share *sysconfdir* among different hosts. That's why often the configuration files are called `toolname/HOSTNAME.conf`, so that different configuration files for different hosts can be specified.

Configuration files are text based and use a very simple and human-readable syntax. The META file shown in listing 1 is an example of the syntax used for the configuration of ISGTC scripts.

3.9 Administrator profiles

In our group 13 persons have the root password. In order to trace who did what change to the system, such as creating a new user, there is a login utility called `isgtc_login`, that is started in the root profile. Every time an administrator logs in as root, he identifies himself by typing his username. This causes environment variables to be set, so that ISGTC scripts can log who did what.

As an added bonus, each administrator can configure its root environment, including shell and editor. I am really awkward when I type in a `tcsh` of a colleague with vi-like key bindings, but on the other hand he works better with it, so it is really nice that we can have it both ways.

3.10 Programming language and style

ISGTC tools are written in one common language, which is in our case Perl. Having only one programming language makes understanding the code for all the people involved in writing and maintaining the scripts easier. It also makes code sharing between modules simple. Of course, if for some reason another language is required, then that language is used.

We did also define how the code should look like, so that it is easier to read and consistent throughout. We did choose the style as defined in the `perlstyle` man-page.

3.11 Testing

The `isgtc_install` script can automatically run test scripts before it installs a given module. With this you can make sure that for example a configuration parsing module always parses all test cases correctly. If there is a test script and it fails, the module isn't installed. This can be very useful when extending a library: you can guarantee that your changes are not going to affect the existing tools based on it. The quality of the whole ISGTC is improved by such automatic tests.

4 Some ISGTC tools

In this paper I mainly wanted to show you the benefits of having a *policy* on administration scripts and maybe convince you to take the ISGTC policy as a basis to write your own.

Something else that might be useful to you are the scripts themselves. We have written more than 100 scripts over the past 4 years and use them on a daily basis to solve all kinds of system administration tasks. All of these scripts are free software released under the GNU General Public License and can be downloaded from the ISGTC home-page.

Although most tools are configurable and are made *generic*, they might make some assumptions about the environment they run in. For example, we use the NIS+ directory service everywhere, so many tools assume that. Therefore some tools might just be perfect as-is for you, some might require some modifications, and some might be only an inspiration for a new tool of yours.

In this section I will present a selection of ISGTC tools, which I find particularly interesting. Have a look at the ISGTC home-page for the full list.

4.1 accountmgr

`accountmgr` is a tool to create accounts using a configuration file that specifies the needed information. It takes care of creating the passwd entry (currently in NIS+), creates the home-directory, sets file-system quotas, and creates email aliases.

The biggest installation we manage has currently 3068 accounts. In order to have a grasp on all these accounts, we need to know more than the first and last name of the user.

Every account that we create belongs to a specific group or customer, which we call *lab*, and is classified by type: `syst` for system accounts, `stud` for students, `staff` for employees, `proj` for projects and group accounts, `guest`, and `ueb` for anonymous accounts used for exercises. According to the type, additional information is required. For `proj` accounts you must, for example, define an *expiry date*, so that you are forced to actively renew it, and a *contact* address, so that the owner of the account can be reached.

Listing 2: Gecos field with additional information

```
    1    2    3         4      5      6            7
Example,proj,nari,2004/04/26,*,2005/12/31, Joe User <joe@ethz.ch>

all types:
  1: title    2: type    3: lab    4: creation date    5: quota
proj-specific:
  6: expiry date    7: contact
```

The authoritative source of information for accounts is the passwd table and we use the gecos field to preserve all the specified information. Listing 2 shows an example gecos field for a project account.

4.2 dbaccmgr

`dbaccmgr` is a tool to create database accounts. The authoritative information about the accounts is stored in the configuration of `dbaccmgr` under *sysconfdir*. `dbaccmgr` reads what should be present on the system and creates any missing accounts. That way we have a documentation about the creation date, the administrator that created it, for what project it is used and who is responsible for it.

When I want to create a new database, I do the following:

1. Add a line for the database in my CVS copy of `etc/dbaccmgr/servername.conf`
2. CVS-commit the change
3. CVS-update `/usr/isgtc/etc/dbaccmgr/servername.conf`
4. Run `dbaccmgr`

This might seem a lengthy process but it actually takes less than 5 minutes. The documentation advantage is well worth the effort.

4.3 homemover

With `homemover` you can move users' homes from one directory to another local or remote directory. Moving a home-directory is pretty complex, because lots of small things need to be fixed after the move has been made, so that the whole system remains in a consistent state.

`homemover` solves the problem that many things can go wrong when moving homes, by first analyzing what there is to do, making a plan of what needs to be done and asking the user if it is OK to proceed. This plan is afterward used not only for deciding what to do, but also to bring back the system to a sane state in case of an error, by following the plan backwards.

Listing 3: homemover in action

```
root(ds)@tardis# homemover tardis:/usr/tardis/home-a/dws \
  tardis:/usr/tardis/home-b/dws

Due actions:

 1. HOMELINK_RM  /home/dws
 2. NISAUTO_RM   /home/dws
 3. LOCAL_CHMOD  /usr/tardis/home-a/dws 0
 4. LOCAL_COPY   /usr/tardis/home-a/dws /usr/tardis/home-b/dws
 5. LOCAL_CHMOD  /usr/tardis/home-b/dws 755
 6. LOCAL_VERIFY /usr/tardis/home-a/dws /usr/tardis/home-b/dws
 7. HOMELINK_ADD /home/dws      /usr/tardis/home-b/dws
 8. NISAUTO_ADD  /home/dws      tardis:/usr/tardis/home-b/dws
 9. LOCAL_REMOVE /usr/tardis/home-a/dws

10. QUOTACHECK /usr/tardis/home-a
11. QUOTACHECK /usr/tardis/home-b

Proceed? [no]
```

For example, the size of the old and new home-directory is checked to be the same at the end. If it isn't, everything is put back as it was before, so that the problem can be safely analyzed and a retrial can be done later.

Listing 3 shows `homemover` asking for confirmation. `HOMELINK_ADD/RM` is for adding or removing sym-links in `/home` and `NISAUTO_ADD/RM` is for adding or removing automounter entries in NIS+ (we use both: sym-links on the servers and automounted homes on the clients).

4.4 isgsnap

Solaris 8 04/01 introduced file-system snapshots. It was finally possible to make reliable backups without bringing the system down to single user mode. That's really great, but there is a catch: creating snapshots is pretty complicated and difficult to automate for doing backups.

`isgsnap` is a script that takes care of all the snapshot creation details on Solaris:

- Suspends special processes that might be locking the file-system.
- Automatically chooses a disk to use as *backing store* for saving all changes to the disk since the snapshot was created.
- Creates a file with the PIDs of the processes using the snapshot and removes the snapshot only if no one else is using it, so that it is possible, for example, to do a backup dump and at the same time create a mirror of the disk.

Listing 4: Example /etc/disks.conf

```
*** partitions ***

# MOUNT-POINT   SIZE    TYPE(OPTIONS)

+ c0t0d0

/               4000    ufs(logging)
swap            4000    swap(nomount)
/scratch        free    ufs(logging)

*** permissions ***

/scratch        1777    root:root

*** exports ***

/scratch        nfs(nosuid,rw=login-01:nova)
```

4.5 diskadm

Like I said, systems should be documented and it would be nice to have accountability to find out who did what, when. What about the disks configuration such as partition tables, mount points, mount options, and the exported shares?

`diskadm`'s job is to configure disks according to a configuration file called `/etc/disks.conf`. The configuration file specifies how each disk *should* be configured. `diskadm` compares this *target* state to the *current* state and builds up a list of actions required to bring the two in sync. We tried to put all the complex code in this planning phase, which will be seen by the user when asking for confirmation, so that it is very safe to use.

Listing 4 shows an example configuration and listing 5 shows `diskadm` asking for confirmation to proceed according to the displayed plan of action.

Besides documenting and making it easier to configure disks, `diskadm` is also useful with *disk-less clients*. You can have hundreds of machines, with disks that need to be partitioned according to your wishes. For such situations there is a `--batch` option that disables asking for confirmation.

4.6 mirror_root

RAID-1 mirroring of the root disk is really a good idea: if the root disk fails you will spare your users a long down-time. That's the theory anyway. What if it is not an hardware failure but a human mistake? Let's say you update your system and some library breaks so badly, that the system crashes and you can't boot it anymore. Your RAID-1 mirror is not going to help you because all the changes were dutifully replicated to the other disk.

Listing 5: Example diskadm session

```
# diskadm
Due actions:

1. PARTITION c0t0d0
   from:
   0: |000000000000000                                   |     0M - 10000M
   1: |              111111111111111111111111111111111111| 10000M - 34730M
   2: |22222222222222222222222222222222222222222222222222|     0M - 34730M

   to:
   0: |000000000000000                                   |     0M - 10000M
   1: |              111111111111111                     | 10000M - 20000M
   2: |22222222222222222222222222222222222222222222222222|     0M - 34730M
   3: |                            3333333333333333333333| 20000M - 34730M

2. FORMAT c0t0d0s1

3. FORMAT c0t0d0s3

4. FSTAB_CHANGE /dev/dsk/c0t0d0s1
   - /dev/dsk/c0t0d0s1 /dev/rdsk/c0t0d0s1 /scratch ufs 2 yes -
   + /dev/dsk/c0t0d0s1  -                 -       swap - no  -

5. FSTAB_ADD /dev/dsk/c0t0d0s3
   + /dev/dsk/c0t0d0s3 /dev/rdsk/c0t0d0s3 /scratch ufs 2 yes -

Proceed? [no]
```

`mirror_root` is a disk mirroring tool, which copies one disk to another, including the partition table. We run it every week on early Monday morning, so that not only we can use it to recover from hardware failures, but we can also use it to recover from fatal mistakes that we might do during the week.

4.7 patchfetch2

Currently we install on our Solaris machines 274 patches. The process of selecting, testing and installing requires a lot of time, also because of this huge number. We did write `patchfetch2` to help us manage them efficiently.

The most interesting aspect of the script is its user interface: the selection of the patches is done with an editor. `patchfetch2` produces a list of patches that are appropriate for the packages that you have installed on your system and starts an editor, where you can mark the patches that you want to download.

Every patch has a lot of information stored in a README file, that might be interesting to read in order to decide if you want to install it. Therefore we use a *folding* editor, that can collapse many lines into one, just showing the first. The patch list contains one line per patch, with the README collapsed under each patch.

4.8 tetre2

We manage about 300 Unix machines. If our name-server changes, we don't want to go and fix `/etc/resolv.conf` manually on every one of them.

`tetre2` (Template Tree II) is our solution to this problem. All changes to the system are grouped by functionality. For example, all changes that are required in order to configure a Postfix client are put together in a single *feature* named `postfix_client-1.0-ds`. You can think of features as a sort of specialized packages for system changes. The features themselves can expose parameters: for example, the mentioned Postfix client feature needs two parameters, the mail domain and the hostname of the mail server.

A central configuration file specifies which features have to be installed on which machines. In the same file the features also are parametrized according to the specific needs. We can configure a group of machines which uses a certain mail server and another group that uses another. All the configuration steps are executed by creating a `cfengine` file, that is then applied to each machine.

There is a lot more to Template Tree II than what I described here. You can find more about it at: `http://isg.ee.ethz.ch/tools/tetre2`

4.9 Windows tools

The ISGTC is designed to be platform agnostic: it allows the integration of administration scripts for all the platforms that we manage. All the scripts we use for system administration on Windows are also integrated in the ISGTC. We install the Windows scripts so that they are available through a Samba share. They are configured with the UNC path of the share instead of `/usr/isgtc/etc`.

Of particular importance are `usermgr` and `hostmgr`, which create users and windows hosts in the Active Directory according to a configuration file.

`bootmgr` is a boot script management tool, similar to System V init scripts, with the difference that with a single configuration file we can specify what machines should run what scripts. We use it to install hotfixes, tighten permissions, fix the PATH environment variable, ...

Further information

On the ISGTC home-page you can find the ISGTC policy document and all our scripts:

```
http://isg.ee.ethz.ch/tools/isgtc
```