# Centralized System Monitoring With Swatch

Stephen E. Hansen and E. Todd Atkins

*Electrical Engineering Computer Facility*
*Stanford University*
*hansen@Sierra.Stanford.EDU*
*atkins@EE-CF.Stanford.EDU*

## Abstract

With the exception of login failures, few systems log the kinds of information that might indicate a security related probe or attack. Even when this information is logged, it is often hidden away in places that are either not monitored regularly or are susceptible to deletion or modification by a successful intruder. To address the problem, our approach begins with the modification of certain system programs to enhance their logging capabilities. A system administrator must often monitor several, perhaps dozens, of systems. Our approach calls for the logging facilities on each of these systems to be configured in such a way as to send a copy of the security and other critical system information to a secure, central logging host system. As one might expect, this central log can see as much as a megabyte of data a day. To keep a system administrator from being overwhelmed by a large quantity of data we have developed an easily configurable log file filter/monitor, called *swatch*. *Swatch* monitors log files and acts to filter out unwanted data and take one or more user specified actions (ring bell, send mail, execute a script, etc.) based upon patterns in the log.

## 1.0 The Problem

It is an unfortunate fact that most Unix® systems, as delivered, do little to ease the job of the security conscious system administrator. Even twenty after its introduction years it is still common to find Unix systems shipped with default configurations that allow easy access to an intruder or that have insecure permissions on critical files. A good system administrator will spend the time and effort needed to find and fix these problems as part of the installation process. But the job of system security doesn't stop there, since security, like every other part of a system administrator's job, is really a never ending task.

What every good system administrator tries to do is keep an eye on the health of each of the systems in his or her care. The health of a system should be reflected in the log messages generated by the kernel and the various daemons and utilities. These messages should also include information relevant to system security. With most systems we have seen, the system's log information is not generally made available to the system administrator in a way that is either secure or convenient. The Unix syslog facility, regardless of the original intent, has traditionally been used as more of a debugging aid than as a tool for system management. The assumption seems to be that system log files are only to be consulted after the fact, to help determine what happened rather than what is happening or what is about to happen.

All modern Unix systems do some logging. This usually consists of memory, disk, or tape errors, jammed ethernet notices, and the like. With the exception of login failures, few systems log the kinds of information that might indicate a security related probe or attack. Even when this information is logged, it is often hidden away in places that are either not monitored regularly or are susceptible to deletion or modification by a successful intruder.

## 2.0 Improved Security Logging

For purposes of monitoring systems security, standard Unix logging features prove to be inadequate and/or inconvenient. To address this problem, our approach begins with the modification of certain system utilities to enhance the reporting done, particularly with regard to possible security related activities. Table 1 lists some of the utilities modified and the changes made to their logging capabilities.

**TABLE 1. List of logging enhancements made to several system programs**

| Program | Logging Enhancements |
|---|---|
| *fingerd:* | Reports the originating host and the finger target(s) to syslog. |
| *ftpd:* | Reports originating host to syslog. Reports file transfers to a local log file along with the local user name and, if the user is "anonymous", the password. |
| *ruserok:* | Used by *rshd* and *login* when called by *rlogind.* Disallows and reports to syslog any attempts to use a */etc/hosts.equiv* or *~/.rhosts* file that contains a '+'. |
| *rshd:* | Reports the access status, local user, remote user and host, and the command issued to a local log file. |
| *login:* | Number of tries reduced to three. Reports to syslog on 'INCOMPLETE LOGIN ATTEMPT', 'REPEATED LOGIN ATTEMPT', and 'ROOT LOGIN REFUSED'. Includes the account names attempted and the originating host. |

At our site we were fortunate enough to have access to the vendor's source code for all our utilities. While this is not possible for everyone, each of the utilities listed in Table 1 are available from various network archive sites. In a few cases it might be preferable to use the public version instead so as to improve portability. Another source of security related information is from the tcp wrapper code written by Weitse Venema[1]. Besides providing access control for those network services run out of *inetd*, it generates information via syslog about the connections it mediates.

One important utility not listed in Table 1 is *sendmail*. Even without modification *sendmail* can be configured to generate a plethora of status information. Unfortunately, *sendmail* isn't very discriminating in what it reports, assigning every status message the same priority. Modifying *sendmail* to assign appropriate priority levels to its status messages is on our to-do list.

## 3.0 Centralized Logging with syslog

When we have added to the logging capabilities of the various utilities, we have, for the most part, made use of the syslog[1] library functions. Besides providing a consistent and relatively stan-

dard logging interface, syslog directs logging messages to different files or hosts based upon the source of the message and its level of importance.

The way our facility is set up, each server system keeps its own copy of most of the syslog messages in the file */var/log/syslog*. Syslog files are rotated on a daily basis, compressed, and kept online for about a week. Log messages that might reflect a system's health or potential security problems are also forwarded to a central log host, the **logmaster**. In practice this means that almost everything except *sendmail* status messages are sent to the logmaster. Leaving the *sendmail* status messages on the servers cuts down on the network traffic due to syslog without significantly affecting our ability to monitor. On our systems the *sendmail* messages can account for as much as 90% of a host's log messages, although 50% is more common. Appendix A shows the syslog configuration file (*/etc/syslog.conf*) for a host being monitored. The last three lines in the file are responsible for sending data to the logmaster.

Copying the syslog information to a central site is done for several reasons. First, it provides security and redundancy. If the log files on the originating host are destroyed or modified, either accidently or by malicious intent, those on the hopefully more secure central site will be left intact. Second, it simplifies the monitoring of all the log information. By collecting information from a number of systems in a single ordered file, problems may be found that would be missed if viewed in isolation. For example, a single failed log in attempt on one system might be attributed to a typing error. The same failed log in attempt occurring on several systems in sequence could indicate an intruder trying to break in. Collecting information from several different system utilities as well as from more than one system can provide information indicating a pattern of attack. Several fingers followed by a failed *login* or *rsh* command is a common pattern revealed by this type of monitoring. Logging the target of the *finger* requests has been extremely useful in exposing undue interest or, in the case of multihopped fingers, attempts to hide a cracker's staging area.

## 4.0 Winnowing the Chaff – An Introduction to *Swatch*

Our facility manages about a dozen file and CPU servers which have over 50 client machines. The server systems receive an enormous amount of log information through the syslog daemon. Even after filtering out the *sendmail* information messages the logmaster sees about a megabyte of syslog messages per day. As one can imagine, sorting through that much information on a daily basis can be very time consuming. We also found that some important log entries tend to get lost among all of the less important entries when one examines the log files.

One solution to this problem would be to search for certain types of information, which can be done by using the *egrep* program with some complex command line arguments. Even with this solution one still has the problem of having to constantly monitor the output so that the urgent information is seen when it comes in. Some of this information needs to be acted on soon after it is received. For example, if the kernel on a file server machine dies then somebody needs to be alerted so the machine can be brought back up as quickly as possible. For us the most desirable solution was to have a more complex program weed through the log and do a few simple tasks when certain types of information were found. We decided to call this program *swatch,* which stands for *S*imple *WATCH*er.

---

1. For those systems with older 4.2BSD style syslogs that do not support non-local logging, a 4.3BSD version is also available in the BSD sources on several of the net archives (i.e. ftp.uu.net or gatekeeper.dec.com).

## 4.1 *Swatch* Design Goals

There were four goals that were set when designing *swatch*.

1. Configure the program in such a way that it would only take a few minutes to teach any systems administrator how to use it.
2. Have a simple set of actions that could be performed after receiving certain types of information.
3. Allow the users to define their own actions if they like, and allow them to use parts of the input as arguments to the action.
4. Once *swatch* is running it should be reconfigurable on demand or after a specified interval without having to stop and restart the program by hand.

## 4.2 Using *Swatch*

*Swatch* may be run three different ways: make a single pass through a file; look at messages that are being appended to a file as that file is being updated; and examine the standard output of a program. A complete description of *swatch*'s command line options can be found in Appendix B.

*Swatch*'s most powerful function is in examining information as it is being appended to a log file. We use *swatch* to look at messages as they are being added to the syslog file, alerting us immediately to serious system problems as they occur. Using a *tail*(1) of /var/log/syslog is the default action for *swatch* but another file can be "tailed" by using the **-t** command line option as in

```
swatch -t /var/log/authlog
```

Receiving timely notification of certain types of probes or attacks often enables us to find out which users are logged on to the originating system. Finding out such information can help identify hackers or compromised accounts.

Using the **-f** option, *swatch* can be made to read in and process a file from beginning to end. This single pass feature can be used to examine old syslog or other text files.

```
swatch -f /var/log/syslog.0
```

This option can be used to catch up on the contents of log files after being away from the computer for a while (like after vacationing in Hawaii for a week). This feature is often used to filter through several megabytes of old syslog files to look for evidence of suspected system and network related problems as well as system probes and break-in attempts.

Having *swatch* examine the output from a program is also useful. For example, one might want to sort through process accounting or other audit information that is not kept in a plain text file and requires special processing to read.

```
swatch -c swatchrc.acct -p lastcomm
```

## 4.3 Implementation

*Swatch* relies heavily on expression matching. For this reason the Perl[2] language was used because of its Awk and C like characteristics, as well as its increasing familiarity among systems administrators.

*Swatch* has three basic parts: a configuration file, a library of actions, and a controlling program.

### 4.3.1 Configuration File

Each non-comment line in a *swatch* configuration file consists of two tab separated fields: a pattern expression and a set of actions to be done if the expression is matched. A line's pattern field consists of one or more comma-separated expressions while the action field may contain one or more comma-separated actions.

/pattern/[,/pattern/,…] action[,action,…]

The patterns must be regular expressions which Perl will accept, which are very similar to those used by the Unix *egrep* program. Each string to be matched is compared, in order, with the expressions in the configuration file and if a match is found the corresponding actions are taken. A copy of the Unix manual page for *swatch*'s configuration file is listed in Appendix C.

Lines beginning with the '#' character are treated as comment lines and are ignored.

### 4.3.2 Actions

*Swatch* understands the following actions: echo, bell, ignore, write, mail, pipe, and exec.

- The echo action causes the line to be echoed to *swatch*'s controlling terminal. An optional mode argument causes the text to be shown in normal, bold, underscore, blinking, or inverse mode. Normal mode is the default.
- The bell action sends a bell signal (^G) to the controlling terminal. An optional argument specifies the number of bell signals to send, with one being the default.
- The ignore action causes *swatch* to ignore the current line of input and proceed to the next one. The ignore action is mainly useful early on in the configuration file to filter out specific unimportant information that would otherwise match a more general expression found later in the configuration file.
- The write and mail actions can be used to send a copy of the line to a user list via the write and mail commands.
- The pipe and exec actions were added to provide some flexibility. The pipe action allows the user to use matched lines as input to a particular command on the system. The exec action allows the user to run a command on the system with the option of using selected fields from the matched line as arguments for the command. A *$N* will be replaced by the *Nth* field in the matched line. A *$0* or a *$** will be replaced by the entire line.

See Appendix C for more details on the actions and their arguments.

### 4.3.3 Controlling Program

The controlling program is *swatch,* but the real work is done by a watcher process. *Swatch*'s first task is to translate the configuration file into a Perl script. After creating the watcher script, *swatch* forks and executes it as the watcher process. The watcher script also contains a signal handler that is called after receiving a terminate signal, SIGTERM, which attempts to clean up and then exit.

## 5.0 Examples

We have previously described several ways in which *swatch* can be used. In this section we will illustrate the two most common ways in which *swatch* is used at out facility. First, we have a *swatch* job running continuously looking for failed login attempts and system crashes and reboots. The *swatch* configuration file we use for this purpose is shown in Figure 1. A portion of *swatch*'s output generated by using this configuration file is shown in Figure 2.

Second it's common for each system administrator to have a customized *swatch* configuration file in his or her home directory, *~/.swatchrc,* that contains pattern/action pairs that are personally interesting, or that pertain to his or her system responsibilities. A *swatch* job using this configuration file is generally run in a window while the administrator is logged in. The personal *swatch* configuration file of one of the authors is shown in Figure 3, while Figure 4 shows an hour's output generated by this script.

## 5.1 Example 1. Constant monitoring for high priority events.

This *swatch* script runs continuously looking for high priority events, such as failed login attempts that might indicate an attempted break-in, and system panics and reboots. The first non-comment line looks for a */bin/login* syslog message of the form

```
Jul 30 13:49:47 Sierra login: REPEATED LOGIN FAILURES ON ttyq0 FROM cert.cert.org:
root, anonymou, anonymou
```

The string REPEATED matches the pattern and *swatch* echoes the line and three bells to stdout, mails a copy of the line to the user who is running *swatch*, and then executes a script to finger the host that initiated the failed login.

The next pattern looks for messages from the machine room temperature monitor. The rest of the pattern/action lines in the configuration file look for panic, halt, or reboot messages from various systems. If any of these patterns are matched *swatch* will take the same actions as for the invalid login. However, instead of a backfinger script it will execute a script to call a pager with a

running *swatch* with this configuration file.

**FIGURE 1. *Swatch* configuration file for continuous monitoring**

```
#
# Swatch configuration file for constant monitoring
#
# Bad login attempts
/INVALID|REPEATED|INCOMPLETE/ echo,bell=3,exec="/eecf/adm/bin/badloginfinger $0"

# Machine room temperature
/WizMON/                      echo=inverse,bell=3

# System crashes and halts
/(Gordon-Biersch|Anchor)/&&/(panic|halt)/echo,bell,mail,exec="call_pager 3667615
 0911"
/(isl|coffee)/&&/(panic|halt)/  echo,bell,mail,exec="call_pager 3667615 1911"
/Sierra/&&/(panic|halt)/        echo,bell,mail,exec="call_pager 3667615 2911"
/(gloworm|stjames)/&&/(panic|halt)/     echo,bell,mail,exec="call_pager 3667615
3911"
/(osiris|shemesh)/&&/(panic|halt)/      echo,bell,mail,exec="call_pager 3667615
4911"
/(panic|halt)/                          echo,bell,mail

# System reboots
/(gloworm|stjames)/&&/SunOS Release/    echo,bell,mail,exec="call_pager 3667615
3411"
/(osiris|shemesh)/&&/SunOS Release/     echo,bell,mail,exec="call_pager 3667615
4411"
/(Gordon-Biersch|Anchor)/&&/SunOS Release/      echo,bell,mail,exec="call_pager
3667615 0411"
/Sierra/&&/SunOS Release/               echo,bell,mail,exec="call_pager 3667615
2411"
/(isl|coffee)/&&/SunOS Release/ echo,bell,mail,exec="call_pager 3667615 1411"
/SunOS Release/                 echo,bell,mail
```

**FIGURE 2. Output from *swatch* running the configuration file in Figure 1 over a 24 hour period**

```
Caught a SIGHUP -- restarting
Jul 30 13:49:03 Sierra login: REPEATED LOGIN FAILURES ON ttyq0 FROM sn01.sncc.lsu.edu:
guest, guest, guest
Jul 30 13:49:47 Sierra login: REPEATED LOGIN FAILURES ON ttyq0 FROM sn01.sncc.lsu.edu:
root, anonymou, anonymou
Jul 30 13:54:57 Sierra login: REPEATED LOGIN FAILURES ON ttype FROM McCulloughA+70:
daniels', aniels, danielss
Jul 30 15:15:32 osiris login: REPEATED LOGIN FAILURES ON ttyp5 FROM Epi: berg, vt100,
^G^Hf
Caught a SIGINT -- shutting down
```

## 5.2 Example 2. Individualized *swatch* configuration file

Individuals may design customized *swatch* configuration files that look for patterns and take appropriate actions depending on their personal preferences. The configuration file shown in Figure 3 is run in a workstation window whenever the system administrator is logged in. The output is generally ignored or only occasionally glanced at unless the bell alerts him or her to a message of interest. Note that the tftpd pattern/action lines in Figure 3 ignore *tftp* requests from valid hosts and alert the user to invalid requests.

**FIGURE 3. Personalized *swatch* configuration file**

```
#
# Personal Swatch configuration file
#

# Alert me of bad login attempts and find out who is on that system
/INVALID|REPEATED|INCOMPLETE/                    echo=underline,bell=3

# Important program errors
/LOGIN/                                          echo=inverse,bell=3
/passwd/                                         echo=bold,bell=3
/ruserok/                                        echo=bold,bell=3

# Ignore this stuff
/sendmail/,/nntp/,/xntp|ntpd/,/faxspooler/       ignore

# Report unusual tftp info
/tftpd.*(ncd|kfps|normal exit)/                  ignore
/tftpd/                                          echo,bell=3

# Kernel problems
/(panic|halt|SunOS Release)/                     echo=blink,bell
/file system full/                               echo=bold,bell=3
/vmunix.*(at|on)/                                ignore
/vmunix/                                         echo,bell

# fingers of root, guest, or myself, or coming from a terminal server (tip) are
interesting.
/fingerd.*(root|[Tt]ip|guest)/                   echo,bell=3
/atkins/                                         echo=inverse,bell=3
/su:/                                            echo=bold

# echo whatever is left.
/.*/                                             echo
```

**FIGURE 4. Output from *swatch* using the configuration file in Figure 3 over the course of an hour**

```
Jul 30 14:01:58 Sierra fingerd[1179]: sunrise.Stanford.EDU (36.93.0.20.3842) ->
"sheppard"
Jul 30 14:02:17 isl fingerd[349]: alice.Stanford.EDU (36.12.0.202.3476) -> "ju"
Jul 30 14:02:21 Sierra fingerd[1194]: alice.Stanford.EDU (36.12.0.202.3477) ->
"chung"
Jul 30 14:06:42 farm.Stanford.EDU fingerd[9212]: elaine9.Stanford.EDU
(36.21.0.125.2192) -> "lan"
Jul 30 14:07:43 Sierra fingerd[1452]: ee.technion.ac.il (132.68.48.3.2223) -> "boaz"
Jul 30 14:09:32 java.Stanford.EDU last message repeated 2 times
Jul 30 14:09:57 Sierra login: REPEATED LOGIN FAILURES ON ttype FROM McCulloughA+70:
daniels', aniels, danielss
Jul 30 14:10:07 Gordon-Biersch fingerd[15264]: EE-CF (36.2.0.107.1302) -> ""
Jul 30 14:10:36 Sierra fingerd[1570]: unstable.Stanford.EDU (36.59.0.12.9521) ->
"ruff"
Jul 30 14:14:14 Sierra fingerd[1660]: leland.Stanford.EDU (36.21.0.69.3474) ->
"pohalski"
Jul 30 14:24:34 Sierra fingerd[1853]: N2.SP.CS.CMU.EDU (128.2.250.82.3480) -> "cclee"
Jul 30 14:25:07 Sierra fingerd[1857]: ee.technion.ac.il (132.68.48.3.2229) -> "boaz"
Jul 30 14:26:16 isl fingerd[1384]: elaine22.Stanford.EDU (36.21.0.210.1480) -> ""
Jul 30 14:31:16 isl fingerd[2006]: elaine22.Stanford.EDU (36.21.0.210.1482) ->
"abbas"
Jul 30 14:31:39 eindhoven.Stanford.EDU fingerd[25244]: elaine22.Stanford.EDU
(36.21.0.210.1481) -> ""
```

```
Jul 30 14:34:26 isl fingerd[2200]: Sunburn.Stanford.EDU (36.8.0.178.1376) ->
"stokesberry"
Jul 30 14:34:37 isl fingerd[2204]: Sunburn.Stanford.EDU (36.8.0.178.1377) -> "eric"
Jul 30 14:35:21 isl fingerd[2217]: Sunburn.Stanford.EDU (36.8.0.178.1380) -> "eric"
Jul 30 14:38:13 isl su: 'su root' succeeded for craig on /dev/ttyq3
Jul 30 14:40:05 isl su: 'su marcg' succeeded for jackk on /dev/ttys0
Jul 30 14:40:07 Sierra fingerd[2323]: ee.technion.ac.il (132.68.48.3.2236) -> "boaz"
Jul 30 14:40:13 isl su: 'su root' succeeded for jackk on /dev/ttys0
Jul 30 14:40:47 isl fingerd[2479]: elaine22.Stanford.EDU (36.21.0.210.1488) ->
"bednarz"
Jul 30 14:41:04 isl fingerd[2492]: Sunburn.Stanford.EDU (36.8.0.178.1413) -> "baas"
Jul 30 14:41:10 coffee su: 'su root' failed for craig on /dev/ttyp3
Jul 30 14:41:14 coffee su: 'su root' succeeded for craig on /dev/ttyp3
Jul 30 14:41:19 isl fingerd[2499]: Sunburn.Stanford.EDU (36.8.0.178.1415) -> "bevan"
Jul 30 14:42:34 Sierra fingerd[2387]: macro.Stanford.EDU (36.59.0.131.1301) ->
"gdmiller"
Jul 30 14:53:22 isl fingerd[3182]: rascals (36.60.0.110.2661) -> "hoffmann"
Jul 30 14:55:32 Gordon-Biersch su: 'su root' failed for atkins on /dev/ttypa
Jul 30 14:55:36 Gordon-Biersch su: 'su root' succeeded for atkins on /dev/ttypa
Jul 30 14:56:22 isl ftpd[3313]: connection from ultrasound.Stanford.EDU at Thu Jul 30
14:56:22 1992
Jul 30 14:57:37 osiris vmunix: /eecf: file system full
Jul 30 14:57:37 osiris vmunix: /eecf: file system full
Jul 30 14:58:12 isl su: 'su root' succeeded for atkins on /dev/ttyt9
Jul 30 14:59:20 isl yppasswdd[77]: equitz: password incorrect
Jul 30 15:00:27 isl yppasswdd[77]: siu: password incorrect
Jul 30 15:02:17 isl fingerd[3800]: alice.Stanford.EDU (36.12.0.202.3484) -> "ju"
```

## 6.0 Other useful programs

We have written a few scripts which we have found useful when using the *swatch* package.

## 6.1 Reswatch

*Reswatch* was written to run out of *cron* periodically. It finds all instances of *swatch* that the user is running and sends a SIGHUP. This is useful if *swatch* is getting its input from an active log file, like *syslog*, that is moved and rendered inactive. Since we want to start getting our input from the new active log file, the old file handle needs to be closed and the new one opened. This effect is achieved when *swatch* aborts one script and starts a new one after receiving a SIGHUP.

## 6.2 Badloginfinger

*Badloginfinger* is used to finger the host that generated an unsuccessful login attempt. Output from this command is placed in its own log file. This is most useful when culprits fail to log in to a system using an unauthorized account, like root, guest, or anonymous. Some administrators might be surprised at how often this happens on their systems.

## 6.3 CallPager

For those who must carry a pager, this is very useful for receiving urgent information, such as serious system failures or possible security breaches. This is a simple script which uses the Unix *tip* command to call a pager through a modem and leave a code number to indicate the type of message detected. Users can customize the codes so that they can tell exactly what type of message was detected, and the system it came from.

## 7.0 Conclusions

Over the past six months *swatch* has proven to be a valuable tool for monitoring the health of a large collection of workstations and servers. On several occasions we have been able to detect intruders probing our systems who would probably have been missed without centralized logging and *swatch*. On two occasions it prevented system meltdown when air conditioning units failed late at night. Its value has increased as we have gathered more experience in optimizing the *swatch* configuration file entries.

In the near term, we see a need to improve the logging capabilities of additional system utilities (i.e. *sendmail, ntp, ypserv, xdm*). We plan to gather suggestion from other sites using the package before making substantial changes to *swatch* itself.

## 8.0 Availability

*Swatch* source and documentation along with its companion scripts are available via anonymous ftp from Sierra.Stanford.EDU, [36.2.0.98], in the *pub/sources* directory. Listserver access is available from listserver@Sierra.Stanford.EDU.

## 9.0 References.


1. W. Venema. “TCP WRAPPER, A Tool for Network Monitoring, Access Control, and for Setting Up Booby Traps”, *Proc. 1992 USENIX Security Symposium,* USENIX Association, Sept. 1992.
2. L. Wall and R. Schwatz. “Programming Perl”, O’Reilly and Associates, Sebastopol, CA. 1991.

## Appendix A. A Syslog Configuration File.

```
# syslog configuration file.
#
# This file is processed by m4 so be careful to quote (`') names
# that match m4 reserved words. Also, within ifdef's, arguments
# containing commas must be quoted.
#
# Note: Have to exclude user from most lines so that user.alert
# and user.emerg are not included, because old sendmails
# will generate them for debugging information. If you
# have no 4.2BSD based systems doing network logging, you
# can remove all the special cases for "user" logging.
#
*.err;kern.debug;auth.notice;user.none                  /dev/console
*.err;kern.debug;mail.crit;user.none                    /var/adm/messages
lpr.debug                                               /var/adm/lpd-errs

# You may want to add operator to the following if your operator
# is a traditional Unix style operator.
*.alert;kern.err;daemon.err                             root

*.emerg;user.none *

# for loghost machines, to have authentication messages (su, login, etc.)
# logged to a file, un-comment out the following line and adjust the file name
# as appropriate.
#
auth.notice                                             /var/log/authlog
auth.notice                                             /var/log/syslog
daemon.info                                             /var/log/syslog
mail.debug                                              /var/log/syslog
kern.debug                                              /var/log/syslog

# following line for compatibility with old sendmails. they will send
# messages with no facility code, which will be turned into "user" messages
# by the local syslog daemon. only the "loghost" machine needs the following
# line, to cause these old sendmail log messages to be logged in the
# mail syslog file.
#
user.alert                                              /var/log/syslog

#
# non-loghost machines will use the following lines to cause "user"
# log messages to be logged locally.
#
user.err                                                /dev/console
user.err                                                /var/adm/messages
user.err                                                /var/log/syslog
user.alert                                              /var/log/syslog

# Send most everything to the LogMaster
*.emerg;*.alert;*.crit;*.err;*.warning;*.notice;*.info;mail.none
@logmaster
kern.debug;mail.crit;mail.err
@logmaster
```

## Appendix B. Unix man page for *swatch*

SWATCH(8) MAINTENANCE COMMANDS SWATCH(8)

**NAME**

swatch - simple watcher

**SYNOPSIS**

**swatch** [ **-c** *config_file* ] [ **-r** *restart_time* ]
[[ **-f** *file_to_examine* ] | [ **-p** *program_to_pipe_from* ] | [ **-t** *file_to_tail* ]]

**DESCRIPTION**

**Swatch** is designed to monitor system activity. **Swatch** requires a configuration file which contains *pattern(s)* to look for and *action(s)* to dowhen each pattern is found.

**OPTIONS**

**-c** *filename* Use *filename* as the configuration file.

**-r** *restart_time* Automatically restart at specified time. *Restart_time* can be in any of the following formats:

**+hh:mm**
Restart after the specified time where *hh* is hours and *mm* is minutes.

**hh:mm[am|pm]**
Restart at the specified time.

You may specify only one of the following options:

**-f** *filename* Use *filename* as the file to examine. **Swatch** will do a single pass through the named file.

**-p** *program_name* Examine input piped in from the *program_name*.

**-t** *filename* Examine lines of text as they are added to *filename*.

If **swatch** is called with no options, it is the same as typing the command line

swatch -c ~/.swatchrc -t /var/log/syslog

**SEE ALSO**

swatch(5), signal(3)

**FILES**

/var/tmp/..swatch..PID Temporary execution file

**AUTHOR**

*E. Todd Atkins* (Todd_Atkins@EE-CF.Stanford.EDU)
EE Computer Facility
Stanford University

**NOTES**

Upon receiving a ALRM or HUP signal swatch will re-read the configuration fileand restart. Swatch will terminate gracefully when it receives a QUIT, TERM, or INT signal.

## Appendix C. Unix man page for swatch configuration files



**NAME**

swatchrc - configuration file for the simple watcher swatch(8)

**SYNOPSIS**

**~/.swatchrc**

**DESCRIPTION**

This configuration file is used by the **swatch**(8) program to determine what types of expression patterns to look for and what type of action(s) should be taken when a pattern is matched.

The file contains two TAB separated fields:

/pattern/[,/pattern/,...] action[,action,...]

A pattern must be a regular expression which **perl**(1) will accept, which is very similar to the regular expressions which **egrep**(1) accepts.

The following actions are acceptable:

| | |
|---|---|
| **echo[=mode]** | Echo the matched line. The text mode may be *normal, bold, underscore, blink, inverse*. Some modes may not work on some terminals. **Normal** is the default. |
| **bell[=N]** | Echo the matched line, and send a bell N times (default = 1). |
| **exec=command** | Execute *command*. The *command* may contain variables which are substituted with fields from the matched line. A *$N* will be replaced by the *Nth* field in the line. A *$0* or *$** will be replaced by the entire line. |
| **ignore** | Ignore the matched line. |
| **mail[=address:address:...]** | Send *mail* to *address(es)* containing the matched lines as they appear (default address is the user who is running the program). |
| pipe=command | Pipe matched lines into *command*. |
| **write[=user:user:...]** | Use **write**(1) to send matched lines to *user(s)*. |

**SEE ALSO**

**swatch**(8), **perl**(1)

**AUTHOR**

*E. Todd Atkins* (Todd_Atkins@EE-CF.Stanford.EDU)
EE Computer Facility
Stanford University