

e-script

e-tunity

2000 ff.

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Installation and Configuration</b>	<b>3</b>
2.1	Installation . . . . .	3
2.1.1	Compilation and Installation of the Binary . . . . .	3
2.1.2	Configuration of the TCP service . . . . .	3
2.2	The Client Side . . . . .	4
2.3	The Configuration File . . . . .	4
<b>3</b>	<b>Hints and Tips</b>	<b>6</b>

# Chapter 1

## Introduction

`e-script` is a tool that is used in the development process on Unix servers, where sources and commands are received from a (possibly non-Unix) platform. E.g., the source control may take place on a Windows platform, while the build process and execution takes place on Unix.

`e-script` provides a TCP tunnel through which sources and instructions are sent. `e-script` uses a TCP port to receive these data, and uses a resource file to map pseudo-instructions to real Unix commands.

Upon each request, optionally sources are received, and pseudo-commands are mapped to real Unix commands. These are executed on the Unix host. The sender receives the output of these commands, so that the build process can be remotely monitored. Logically `e-script` can be seen as a "remote make" utility.

## Chapter 2

# Installation and Configuration

This chapter describes the installation and configuration of `e-script`.

## 2.1 Installation

To install `e-script` the following steps are necessary:

- Compilation and installation of the binary program `e-script`,
- Configuration of `e-script` as a TCP service under Unix,
- Configuration of the actions of `e-script` through `e-script`'s resource file `e-script.rc`.

### 2.1.1 Compilation and Installation of the Binary

After obtaining the distribution archive `e-script.tar.gz`, the archive can be unpacked using `tar xvzf e-script.tar.gz`. The archive spills its sources into its separate subdirectory `e-script/`.

The building process is quite simple: a `make install` will do. The build process will install the binary `e-script` into a directory pointed to by the environment variable `$EBINDIR`. The presence of the e-tunity utility library `e-lib` is required during the compilation.

### 2.1.2 Configuration of the TCP service

To allow remote connections for `e-script`, the following steps are necessary (which must be performed by `root`):

- The service for `e-script` is added to `/etc/services`. In this example we'll map the TCP port 8080 to the service. The extra line in `/etc/services` is:

```
1  escript 8080/tcp # e-script service
```

- The service name `escript`, now defined in `/etc/services`, is mapped to an executable via `/etc/inetd.conf`. The addition to `/etc/inetd.conf` is:

```
1  escript stream tcp nowait root /usr/e/bin/e-script e-script
```

- Following the last word on this line, optional flags for `e-script` may be specified. E.g., a `-v` will increase the verbosity, `-r resource` will state which resource to use. (The default resource file is `/usr/e/etc/e-script.rc`.)
- Finally, `inetd` must be restarted (e.g., using `killall -1 inetd`.)

## 2.2 The Client Side

The *client side* which connects to `e-script` must follow a strict protocol when sending pseudo-commands and source components. First, the client must connect to a given port (see above, in our example this is port 8080).

Following the connection, the pseudo-command must be sent, terminated by a `^Z` character (ASCII 0x1a). Optionally, a source component is sent, again terminated by `^Z`.

The server side has then received both components and starts executing. The output is sent back to the client, which must read its connected socket, until EOF. The termination of the TCP connection signals the end of information.

## 2.3 The Configuration File

The default location of `e-script`'s resource file is `/usr/e/etc/e-script.rc`. This default is compiled-in. A different resource can be specified in the invocation via `/etc/inetd.conf`.

The structure of the resource file is:

- A tag (pseudo-command),
- Its expansion to a Unix command.

These items occur together on one line, separated by whitespace. E.g., a very simple line in the resource file may be:

```
1  sample ls -Fla /tmp
```

This specifies that when `e-script` receives a pseudo-command `sample`, the output of `ls -Fla /tmp` is sent back via the TCP tunnel.

Usually a source file will also be transferred, e.g., to be compiled. Consider the following more complex line:

```
1  unpack-and-build-prog  cd /home/user/src; \  
2                          tar xzf $(SRC); \  
3                          cd prog; \  
4                          make; \  
5                          make install
```

Notice the "line continuation": lines that end with a backslash are considered to be continued on the next line. The above statement says that any pseudo-command `unpack-and-build-prog` should be accompanied by data (referred to as `$SRC`). The data component will initially arrive on Unix as a temporary file, but will be taken as a `.tar.gz` archive and unpacked in `/home/user/src`. Assuming that the archive contains a directory tree `prog/`, we `chdir` into `prog/` and build from there.

The tag `SRC` in the above example is the way to address the source component of a request. When `e-script` maps its pseudo-command to a real Unix command, `$SRC` is expanded to an actual filename.

## Chapter 3

# Hints and Tips

When using `e-script` in a real-life situation, the following hints may prove useful:

- Remember that remote execution via `inetd` doesn't provide a full working environment (in contrast to a login shell). This can be circumvented by sourcing `/etc/profile` before any other commands, as in:

```
1  unpack-and-build-prog  . /etc/profile; \  
2                          cd /home/user/src; \  
3                          tar xzf $(SRC); \  
4                          cd prog; \  
5                          make; \  
6                          make install
```

- The effective user to execute Unix commands is defined in `/etc/inetd.conf`. (In the example of section 2, this is `root`). An effective way of allowing a multiple-user setup is to let `root` run `e-script`, but to `su` to a different user depending on each pseudo-command. An example is given below:

```
1  # This should run as user "sample"  
2  unpack-and-build-prog  su - sample -c ' \  
3                          . /etc/profile; \  
4                          cd /home/user/src; \  
5                          tar xzf $(SRC); \  
6                          cd prog; \  
7                          make; \  
8                          make install'
```

A different pseudo-command might obviously `su` to a different user.