

**RUDOLF R.H. DITTRICH'S INTERNET PUBLICATION SERIES  
ON ENGINEERING AND PHYSICAL TOPICS**

**VOLUME 2010/2011**

— **special biennial edition** —

(3rd printing, letter-sized version)

**FREE DOWNLOAD EDITION**

**CONTAINS THE FOLLOWING  
ELEMENTARY ARTICLES:**

Notes on a Simple IPv4 Firewall Script

Elements of C Macros and Preprocessor Scripts  
for Transforming Program Code  
to C

Elementary Remarks on Using Interrupts in  
Real-time Systems

A Simple Text File to Mailbox File Converter  
Script

Multi-line Text Searching and the sed Command

Auxiliary Tools for Visually Checking Verbs Fol-  
lowed by the Gerund or the Infinitive

**ISSN: 1867-6278**

**Copyright © 2010-2011 by Dr. Rudolf R.H. Dittrich,  
Schlesierstrasse 16, D-82024 Taufkirchen, Germany**

RUDOLF R.H. DITTRICH'S INTERNET PUBLICATION SERIES  
ON ENGINEERING AND PHYSICAL TOPICS, VOLUME 2010/2011  
ISSN: 1867-6278

Letter-sized Version  
(also available: European A4-paper-sized version (8.26in x 11.69in))

Copyright © 2010–2011 by  
Dr. Rudolf R.H. Dittrich, Schlesierstrasse 16,  
D-82024 Taufkirchen, Germany

All rights reserved — Alle Rechte vorbehalten

## Table of Contents

|  |    |
|--|----|
| Table of Contents .....  | 3  |
| Disclaimer & Copyright Information .....   | 4  |
| Introduction .....   | 5  |
| Articles in this issue:  |    |
| Notes on a Simple IPv4 Firewall Script .....   | 7  |
| 1 Introduction .....   | 7  |
| 2 Parts of the First Netfilter Script .....  | 8  |
| 2.1 Initial Definitions .....  | 8  |
| 2.2 Deletion of Firewall Rules .....   | 9  |
| 2.3 Default Firewall Policies .....  | 9  |
| 2.4 Kernel Settings .....  | 10 |
| 2.5 Input Chain .....  | 11 |
| 2.6 Output Chain .....   | 11 |
| 2.6.1 Local Host .....   | 11 |
| 2.6.2 DNS Resolution Versus Hosts File .....   | 11 |
| 2.6.3 Reception of Mails .....   | 11 |
| 2.6.4 Secure Reception of Mails .....  | 12 |
| 2.6.5 Sending of Mails .....   | 12 |
| 2.6.6 Secure Sending of Mails .....  | 12 |
| 3 Other Firewall Sections .....  | 12 |
| 4 Continuation of Article .....  | 13 |
| Elements of C Macros and Preprocessor Scripts for Transforming<br>Program Code to the C Programming Language ..... | 14 |
| Abstract .....   | 14 |
| 1 Introduction .....   | 14 |
| 2 Need of Preprocessor Scripts .....   | 15 |
| 3 Procedures .....   | 16 |
| 4 Functions .....  | 16 |
| 5 General Code Transformation Scheme .....   | 18 |
| 6 Summary .....  | 18 |
| RD-Avenue <sup>®</sup> Lab.Note 2011-1:<br>Elementary Remarks on Using Interrupts in Real-time Systems .....       | 19 |
| RD-Avenue <sup>®</sup> Lab.Note 2011-2:<br>Mailbox File Generation .....   | 20 |
| Abstract .....   | 20 |
| 1 Introduction .....   | 20 |
| 2 Structure of Mailbox Files .....   | 20 |

|  |    |
|--|----|
| 3 Program Code.....  | 21 |
| 4 Example.....   | 22 |
| 5 Resulting Mailbox File.....  | 22 |
| 6 Summary.....   | 23 |
| RD-Avenue® Lab.Note 2011-3:<br>Multi-line Text Searching and the <code>sed</code> Command..... | 23 |
| Auxiliary Tools for Visually Checking Verbs<br>Followed by the Gerund or the Infinitive.....   | 24 |
| Abstract.....  | 24 |
| 1 Introduction.....  | 24 |
| 2 Examples of Verbs Usually Followed by the Gerund.....  | 25 |
| 3 An Auxiliary Tool for Displaying Verbs Usually Followed by the Gerund.....                   | 26 |
| 4 Auxiliary Shell Script for Checking Verbs Followed by the Infinitive.....                    | 29 |
| 5 Invocation of Tools.....   | 30 |
| 5.1 Invocation of <code>verbs4gerundFinder.sh</code> .....                                     | 30 |
| 5.2 Invocation of <code>verbs4infinitiveFinder.sh</code> .....                                 | 31 |
| 6 Summary.....   | 31 |

## DISCLAIMER & COPYRIGHT INFORMATION:

THE ARTICLES following this introduction, including all content like, for example, text, graphical representations, pictures, program code, etc., are Copyright © 1996–2011 by Rudolf R.H.Dittrich, D - 82024 Taufkirchen, Germany.

The author hereby disclaims all warranties, either implied or express, relating to the accuracy of the information contained in the articles. It is a common experience that printing errors as well as other errors are always possible, even if the utmost care has been taken to prevent their occurrence. Readers are therefore encouraged to inform the author of any mistakes found in the articles in order to facilitate their due correction in a future edition.

Information in this document is provided solely for educational and entertainment purposes without any claim or guarantee regarding the correctness or uniqueness of the articles' contents or their suitability for any purpose. Although partly tested by the author in his specific computer environment, algorithms or program code presented in this document could cease to work, not work as intended, or even unintentionally cause harm or damage, if being tested or run on a system differing from the one used by the author. Use or adaptation of the algorithms or programs for any particular purpose is therefore discouraged and readers or users of this journal are advised to refrain from using the articles, the program code or any other technical information printed in this journal in the setting of their own systems. The author does not encourage or authorize use of the articles' contents for any such purpose. Should the reader or user of this journal use the algorithms, program code or other information for any such unauthorized and discouraged purpose, they shall indemnify and hold harmless the author against all

claims or damages resulting from their unauthorized use of the articles' contents. Unless authorized by the author, no permission is granted to distribute this document electronically, be it via electronic mail or any other imaginable means, or to store it, or part of it, in a computer or any other data-retrieval system. The names of computer languages and other products cited in this documents may constitute trademarks or registered trademarks that are the property of their respective holders and/or copyright owners. Mention of these products does not imply any endorsement or recommendation of their use or their fitness for any particular purpose.

Any violation of the aforementioned rules constitutes an infringement of the author's copyright.

In order to obtain copies, or any other information pertaining to this journal or any of the articles, you are requested to write to the author's address given on page 2 of this document.

## INTRODUCTION

STARTING 2012, the present document and its articles will be accessible on the author's web page

<http://www.rudolf-rh-dittrich.com/rdmisc.html>

using the download link referring to volume 2010/2011 of the journal.

At the end of 2011, the document's download URL was planned to be [http://www.rudolf-rh-dittrich.com/img/rudolf-rh-dittrich.com\\_RudolfRHDittrich\\_InternetPublicationSeries\\_VOL\\_2010\\_2011\\_3rd\\_printing.pdf](http://www.rudolf-rh-dittrich.com/img/rudolf-rh-dittrich.com_RudolfRHDittrich_InternetPublicationSeries_VOL_2010_2011_3rd_printing.pdf) (A4 paper size) and [http://www.rudolf-rh-dittrich.com/img/rudolf-rh-dittrich.com\\_RudolfRHDittrich\\_InternetPublicationSeries\\_VOL\\_2010\\_2011\\_3rd\\_printing\\_LetterSize.pdf](http://www.rudolf-rh-dittrich.com/img/rudolf-rh-dittrich.com_RudolfRHDittrich_InternetPublicationSeries_VOL_2010_2011_3rd_printing_LetterSize.pdf) (letter-sized version)

Note that in the future, the contents of the web page may change on short notice and may, therefore, contain links to additional volumes.

The articles on several engineering and physical topics, originally planned for the 2010 edition, were found to partly coincide with work on a new customer project. The original 2010 edition was therefore withdrawn from publication and replaced by this special biennial edition for 2010/2011 which contains a selection of elementary level articles on non-sophisticated IPv4 firewalls, and several tools for modifying program code and text files. A new set of successor articles of last year's edition will be published with the 2012 edition and should be available shortly after December 2012.

Taufkirchen, December 30, 2011

Addendum: This second printing contains minor modifications in the wording of two articles and a slightly enlarged disclaimer-and-copyright section. The journal is now available in two paper formats, an A4-sized version and a letter-sized version for North American readers with both versions containing the same articles.

In the third printing, several printing errors and the Internet links referring to the journal were corrected.

Advertisement:

AVAILABLE ON REQUEST:  
APPLIED PHYSICS AND ENGINEERING RELATED  
SOFTWARE DEVELOPMENT AND CONSULTING SERVICES

Visit the author's web pages for additional detail:

[www.rudolf-rh-dittrich.com](http://www.rudolf-rh-dittrich.com)

[www.rudolf-dittrich.com](http://www.rudolf-dittrich.com)

[www.rd-avenue.org](http://www.rd-avenue.org)

Dr. Rudolf R.H. Dittrich, System Software — Development Office for  
Applied Physics and Engineering / Embedded Systems / Web Engineering /  
Software Documentation & Translation Services

Schlesierstrasse 16, D-82024 Taufkirchen, Germany

Email: [info@rudolf-rh-dittrich.com](mailto:info@rudolf-rh-dittrich.com)

Internet: [www.rudolf-rh-dittrich.com](http://www.rudolf-rh-dittrich.com)

RD-AVENUE<sup>®</sup> Research and Development — Your avenue to success!

RD-AVENUE<sup>®</sup> is a registered trademark of Dr. Rudolf R.H. Dittrich in Germany.  
Registration in other countries either pending or reserved.

# Notes on a Simple IPv4 Firewall Script

© 2010–2011 by Rudolf R.H. Dittrich  
D-82024 Taufkirchen, Germany

## Abstract

Although becoming gradually outdated with the introduction of the IPv6 protocol, IPv4 systems are still in widespread use for the foreseeable future. This short article presents no new material but shows how the author was using a simple IPv4 netfilter firewall script to restrict network access to and from his former Linux-based system. Except for a short remark on how to switch off IPv6 network traffic, the new protocol will not be covered in this article.

## 1 Introduction

THE TASK of securing one's computer system usually means that a variety of measures are to be taken like, for example<sup>1</sup>,

- the installation and configuration of antivirus software
- careful configuration of the `inetd` daemon (including, if necessary, configuration of the `hosts.allow` and `hosts.deny` files; etc.)
- proper configuration of network access, especially using correct DNS server entries (see the system's `bind` configuration).
- installation of security related system and application program updates on a regular basis
- configuration of a suitable firewall system
- usage of strong passwords that are not being reused on different systems
- restrictive use of network services (viewing of "reliable" websites only; passphrase entry on

SSH or SSL secured connections only; no storage of passwords on the computer; frequent changing of passwords; etc.)

- establishment of a regular data backup policy
- physical security measures like, for example, locking up the computer in a safe storage room; installation of an effective alarm system in the company's offices; if necessary, enhanced electromagnetic shielding of computer equipment including network cables and wall sockets; entrance restriction to all rooms containing computers; if necessary, no use of wifi-based systems; video surveillance of computers while being used; etc.
- proper training of users; security checks on personnel; and similar measures.

The large number of possible attack scenarios means that there can never be a single measure,

---

<sup>1</sup>This article assumes that a netfilter firewall on a Linux or Linux-like system be used. Configuration of other firewalls or different operating systems are not covered in this article.

however sophisticated, that would guarantee a computer system to be forever immune from outside attackers.

Those wishing to reliably protect their business secrets are perhaps best advised to completely separate their Internet browsing experience from the machine containing their data. The same or more advanced security considerations apply should sensitive government or private data be protected. One simple measure for preventing unauthorized network access to data could consist in using different computers for browsing and working. Alternatively, one could reboot the computer for surfing using a special live system that does not allow hard drive access. Both approaches appear to be capable of enhancing security but will not be further discussed in this article.

The configuration of a firewall system thus constitutes just a small step in the overall system security policy. Depending on the hardware setup and

network topology to be secured, it may also be necessary to correctly setup and configure

- modem software ((A)DSL; analog modems; WiFi; etc.)
- routers (routing tables; router firewalls, if present; etc.)
- network software on the computers that are part of the network (including firewalls)

and much more. It is not possible to cover all possible configuration scenarios in a short article like this one or give a comprehensive treatment of even a single of these topics. In this and the next edition, we are instead presenting parts of two simple netfilter firewall scripts which were used by the author for several years for restricting access to the Internet. In this edition, we will present excerpts of a scripts which was set up for mail retrieval purposes.

## 2 Parts of the First Netfilter Script

The following restrictive netfilter script was designed to partly protect a client machine that was using the POP3 and SMTP protocols for retrieving mails from outside servers without being used for Internet surfing<sup>2</sup>. Note that the script should be started with root privileges.

### 2.1 Initial Definitions

The Bash script starts by defining some shell variables with optionally reading in the name of the network interface used. The interface defaults to *eth0* which can be replaced by any valid interface name as follows: after specifying the command line parameter `-e`, the name of the interface should follow. Using `eth2` instead of `eth0`, we would thus invoke the script with parameters `-e eth2`:

```
#!/bin/bash
ECHOCMD=/bin/echo
FW=/sbin/iptables
FW6=/sbin/ip6tables
EXTIF="eth0"
```

<sup>2</sup>This means that Internet access could be restricted to the mail providers' servers. All non-email related Internet traffic could be blocked. This article's script is thus not suitable for Internet browsing, a topic that will be covered by the enlarged firewall script in next year's edition

```

if [ "$1" = "-e" ]; then
    EXTIF=$2
    shift
    shift
fi
MYIP='ifconfig $EXTIF|\
    head -2|\
    grep inet|\
    sed 's/Bcast.*$//g'|sed 's/^.*://g'|sed 's/[ ]//g'
if [ "$MYIP" = "" ]; then
    echo "WARNING: LOCAL IP ADDRESS UNKNOWN, TEMPORARILY ASSUMING"
    echo "192.168.1.2. YOU MUST RESTART THIS SCRIPT"
    echo "IMMEDIATELY AFTER HAVING STARTED THE INTERNET CONNECTION"
    MYIP=192.168.1.2
fi
$ECHOCMD "USING      IFACE   = $EXTIF"
$ECHOCMD " LOCAL_IP_ADDESS = $MYIP"

```

## 2.2 Deletion of Firewall Rules

Next we are following the standard procedure of deleting all IPv4 and IPv6 firewall rules that might be in place at the time the script is being invoked. For the short time interval between issuing the deletion commands and the steps described in the next section, the user's machine will be unprotected. Those extremely cautiously minded would either use the `ifconfig <interface> down` command to bring down their network or physically disconnect their computer:

```

# Delete all
$FW -F
$FW -X
$FW -Z
$FW6 -F
$FW6 -X
$FW6 -Z

```

## 2.3 Default Firewall Policies

Having deleted all firewall rules, we secure the networking system by setting all IPv4 and IPv6 default policies to DROP. This applies to the INPUT, OUTPUT, and FORWARD chains:

```

# Policy
$FW --policy INPUT DROP
$FW --policy OUTPUT DROP
$FW --policy FORWARD DROP

```

```
$FW6 --policy INPUT DROP
$FW6 --policy OUTPUT DROP
$FW6 --policy FORWARD DROP
```

Provided the netfilter system is working well, network access to and from our system should be blocked now. This means that the machine may safely be reconnected again to the Internet.

## 2.4 Kernel Settings

The next step consists in setting several Linux kernel parameters. These are again basic and well-known settings that are used throughout the Internet by many systems in this or a similar, often extended, manner. Details can be found in the following comment lines:

```
# Disable response to ping:
$ECHOCMD "1" > /proc/sys/net/ipv4/icmp_echo_ignore_all

# Disable response to broadcast requests:
$ECHOCMD "1" > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

# Reject source routed packets.
$ECHOCMD "0" > /proc/sys/net/ipv4/conf/all/accept_source_route

# Next, we disable ICMP redirect acceptance in order to prevent
# the modification of routing tables:
$ECHOCMD "0" > /proc/sys/net/ipv4/conf/all/accept_redirects

# Additionally, we enable protection against bogus error messages:
$ECHOCMD "1" > /proc/sys/net/ipv4/icmp_ignore_bogus_error_responses

# Activate reverse path filtering on all interfaces:
for LocalInterface in /proc/sys/net/ipv4/conf/*/rp_filter; do
    $ECHOCMD "1" > $LocalInterface
done

# Not using a multi-homed host, we are also turning off IP forwarding:
$ECHOCMD "0" > /proc/sys/net/ipv4/ip_forward

# For security reasons, we decide to log "martians", i.e.
# source routed packets, spoofed packets, and redirect packets.
$ECHOCMD "1" > /proc/sys/net/ipv4/conf/all/log_martians
```

For additional kernel parameter settings, the interested reader is referred to the Internet or the literature.

## 2.5 Input Chain

Having set up the basic protective settings described above, we continue with "opening" the firewall in the sense of allowing just a finite number of connections to the hosts we are interested in. These hosts include the local machine, or `localhost`, and a limited number of outside servers (see description of output chain in next sections):

```
# INPUT chain
# We permit localhost inputs (necessary for correct operation of printers,
# X server, etc.)
$FW --append INPUT -i lo -j ACCEPT
# Input pertaining to established connections is accepted:
$FW --append INPUT -i $EXTIF --match state --state RELATED,ESTABLISHED \
    -j ACCEPT
# Invalid input connections should be logged:
$FW --append INPUT -i $EXTIF --match state --state NEW,INVALID -j LOG
```

## 2.6 Output Chain

### 2.6.1 Local Host

```
# We again permit localhost output (necessary for correct operation of
# printers, the X server, etc.)
$FW --append OUTPUT -o lo -s $MYIP -d $MYIP -j ACCEPT
$FW --append OUTPUT -o lo -s $MYIP -d 127.0.0.1 -j ACCEPT
$FW --append OUTPUT -o lo -s 127.0.0.1 -d $MYIP -j ACCEPT
$FW --append OUTPUT -o lo -s 127.0.0.1 -d 127.0.0.1 -j ACCEPT
```

### 2.6.2 DNS Resolution Versus Hosts File

DNS server information should not be trusted in a secure environment. Instead of accessing a remote DNS server for obtaining the numeric IP address of our email provider's POP mail and SMTP servers, we may edit the `/etc/hosts` file and specify the numeric and symbolic server addresses there. In all subsequent subsections, strings like `URLOfMailPOPServer`, `URLOfSecurePOPServer`, `URLOfSMTPServer` and `URLOfSecureSMTPServer` refer to the symbolic server address in `/etc/hosts`. The `grep`, `head` and `awk` commands are then being used for retrieving the known numeric IP address from `/etc/hosts`.

### 2.6.3 Reception of Mails

In case of a single POP mail server, we are first retrieving the server's numeric IP address from the hosts file and use this information to tell the `iptables` command which destination address and port to accept. Note that we are using the POP3 protocol's standard port 110:

```
# Reception of mails
```

```
thisIP='grep URLofMailPOPServer /etc/hosts|head -1|awk '{print $1}''
$FW --append OUTPUT -o $EXTIF -s $MYIP -d $thisIP -p tcp --sport 1024:65535 \
    --dport 110 -j ACCEPT
```

### 2.6.4 Secure Reception of Mails

The secure version of mail retrieval proceeds in exactly the same way with port 110 now being replaced by port 995<sup>3</sup>:

```
popserv = URLofSecurePOPServer
thisIP='grep $popserv /etc/hosts|head -1|awk '{print $1}''
$FW --append OUTPUT -o $EXTIF -s $MYIP -d $thisIP -p tcp --sport 1024:65535 \
    --dport 995 -j ACCEPT
```

### 2.6.5 Sending of Mails

For sending mails using the SMTP protocol, port 25 is usually chosen:

```
# Sending of mails
thisIP='grep URLofSMTPServer /etc/hosts|head -1|awk '{print $1}''
$FW --append OUTPUT -o $EXTIF -s $MYIP -d $thisIP -p tcp --sport 1024:65535 \
    --dport 25 -j ACCEPT
```

### 2.6.6 Secure Sending of Mails

The secure sending of mails is often done using port 465:

```
# Secure sending of mails
smtpserv = URLofSecureSMTPServer
thisIP='grep $smtpserv /etc/hosts|head -1|awk '{print $1}''
$FW --append OUTPUT -o $EXTIF -s $MYIP -d $thisIP -p tcp --sport 1024:65535 \
    --dport 465 -j ACCEPT
```

## 3 Other Firewall Script Sections

We finally turn on logging of all output chain related issues thus making sure that there are log file entries suitable for future analysis.

```
# Turn on logging:
$FW --append OUTPUT -o $EXTIF --match state --state NEW,INVALID -j LOG
```

<sup>3</sup>Readers should note that some providers are using different port numbers than the numbers used in this article. Before experimenting with this or similar scripts, one should always ask one's email provider for the current set of connection data.

The sections above covered the basic port settings for retrieving or sending mails to a single email server only. The approach presented in this article can easily be extended to more than one host as follows: We assume that we want to allow access to  $N$  POP3 servers with symbolic addresses

$$a1, a2, a3, \dots, aN$$

that are all part of the `/etc/hosts` file.

We may then replace the single host example of section 2.6.3 by looping over all servers:

```
for popserver in a1 a2 a3 ... aN; do
    thisIP='grep $popserver /etc/hosts|head -1|awk '{print $1}''
    $FW --append OUTPUT -o $EXTIF -s $MYIP -d $thisIP -p tcp \
        --sport 1024:65535 --dport 110 -j ACCEPT
done
```

The other sections on retrieving and sending mails can be extended in a similar way.

## 4 Continuation of Article

In next year's edition, we will continue with a discussion of the beforementioned script and introduce the second netfilter script that was designed for providing a partly secured netbrowsing environment.

CONTINUED IN THE NEXT EDITION...

# Elementary Elements of C Macros and Preprocessor Scripts for Transforming Program Code to the C Programming Language

Copyright © 2010-2011 by Rudolf R.H. Dittrich  
Schlesierstraße 16, D-82024 Taufkirchen, Germany  
All rights reserved / Tous droits réservés / Alle Rechte vorbehalten

## Abstract

This article presents some macros which enable to write code mimicking in part the look of some programming languages of the 1960s and 70s like, for example, Algol and Pascal. Supplemented by scripts for pre-processing code, this approach is occasionally capable of replacing some non-C language compilers — provided the code is not too complicated and thus easily transformable to C. The presentation is intended to give some hints on how to write such macros and scripts at an elementary level only and is thus neither intended for advanced level courses nor for introductory material on writing or designing C-language programs.

## 1 Introduction

USING PREPROCESSOR macros, we may rewrite C code in a way that occasionally resembles code written in other languages. The following example should get us started. Upon defining

```
#define AND      &&
#define _AND_   &&
#define OR      ||
#define _OR_    ||
#define IF      if(
#define THEN    )
#define BEGIN   {
#define END     }
#define ELSE    else
#define ELSIF   else if
#define _GT_    >
#define _GE_    >=
#define _EQ_    ==
#define _LE_    <=
#define _LT_    <
```

```
#define _NOT_    !
#define INCREMENT(a) a++
```

a typical `if-then` construction, written in C, like

```
if ((a < 3) && (b == c)) {
    a++;
}
```

can be rewritten in the following way:

```
IF (a _LT_ 3) AND (b _EQ_ c) THEN
BEGIN
    INCREMENT(a);
END
```

This example can easily be extended to cover more complicated `if-then-else` clauses. Consider, by way of example, the following elementary C code:

```
if ((a < 3) && (b == c)) {
    a++;
} else if ((a == 9) || (a == c)) {
```

```

    a = 0;
} else {
    a = b;
}

```

Using the beforementioned preprocessor macros, this can also be written as

```

IF (a _LT_ 3) AND (b _EQ_ c) THEN
BEGIN

```

```

    INCREMENT(a);
END ELSIF (a _EQ_ 9) OR (a _EQ_ c) THEN
BEGIN
    a = 0;
END ELSE
BEGIN
    a = b;
END

```

## 2 Need of Preprocessor Scripts

There exist various syntactic constructions which cannot be represented by C preprocessor macros. This means that we may use preprocessor macros for mimicking only part of FORTRAN, Pascal or Algol but not the whole language. By way of example, consider the old FORTRAN syntax of denoting the boolean comparison operators which are given by

| FORTRAN | Meaning          | Corresponding syntax |        |
|---------|------------------|----------------------|--------|
|         |                  | C                    | Pascal |
| .GT.    | greater than     | >                    | >      |
| .GE.    | greater or equal | >=                   | >=     |
| .EQ.    | equal(s)         | ==                   | =      |
| .LE.    | less or equal    | <=                   | <=     |
| .LT.    | less than        | <                    | <      |
| .NOT.   | not              | !                    | NOT    |
| .AND.   | and              | &&                   | AND    |
| .OR.    | or               |                      | OR     |

The period in the FORTRAN operators cannot be part of a valid C macro name, or definition, and thus prevents the use of these definitions in a header file. We may use an additional preprocessor script, however, which would first translate the *period* to an *underscore*. This way we arrive at a file containing intermediate code using the tokens already given in the introduction. The intermediate code may thus be compiled using C preprocessor macros.

An excerpt of a suitable preprocessor script, making use of the `sed` command<sup>1</sup>, could read as follows:

```

cat source.f | \
  sed 's/\.GT\./_GT_/ig' | \
  sed 's/\.GE\./_GE_/ig' | \
  sed 's/\.EQ\./_EQ_/ig' | \
  sed 's/\.LE\./_LE_/ig' | \

```

<sup>1</sup>`sed` is the well-known *stream editor*.

```
sed 's/\.LT\_./\_LT\_/ig' |\
sed 's/\.NOT\_./\_NOT\_/ig' |\
sed 's/\.AND\_./\_AND\_/ig' |\
sed 's/\.OR\_./\_OR\_/ig' > intermediateCode.txt
```

### 3 Procedures

Using the following additional definitions, namely

```
#define STRING      char *
#define PROCEDURE   void
#define FUNCTION(fname,fparams, \
    freturntype) freturntype fname(fparams)
#define WRITELN(a)  printf("%s\n", a)
```

a function like

```
void showString(char *str)
{
    printf("%s\n", str);
}
```

would then be represented by

```
PROCEDURE showString(STRING str)
BEGIN
    WRITELN(str);
END
```

### 4 Functions

Additional macros for modifying type definitions, depicted in the following example, may come in handy if it comes to more complex functions:

```
#define INT          int
#define REAL         float
#define DOUBLE       double
#define CHARACTER    char
```

Consider a simple function for adding two floating point numbers which we would write in C as follows:

```
float addTwoFloats(float a, float b)
{
    return a+b;
}
```

Using the new definitions, function `addTwoFloats` is both equivalent to

```
REAL addTwoFloats(REAL a, REAL b)
BEGIN
    RETURN a+b;
END
```

(1)

and to

```
FUNCTION(addTwoFloats, REAL a, REAL b, REAL)
BEGIN
    RETURN a+b;
END
```

These examples show again the limitations of using C macros only. It is not possible to use such macros to write the beforementioned function in a Pascal-like way as follows or to interpret the original unmodified Pascal function:

```
FUNCTION addTwoFloats(a:REAL; b:REAL):REAL
BEGIN
    addTwoFloats:=a+b;
END
```

(2)

Adding a preprocessor script and making use of regular expressions, the prototype may be transformed as follows<sup>2</sup>:

```
FNAME='cat function.pas|tr '\n' ''|\
    sed 's/^(.*FUNCTION \([^ \t()][^ \t()*)\)(.*\/1/g','
cat function.pas |\
    sed 's/\([A-Za-z0-9_][A-Za-z0-9_]*\):\([^;]*\)/\2 \1/g' |\
    sed 's/FUNCTION[ \t][ \t]*\([^ \t]*\)[ \t]*:[ \t]*'\
'\([A-Za-z0-9_][A-Za-z0-9_]*\)/\2 \1)/g' |\
    sed 's/'"$FNAME"[ \t]*:=/RETURN /g' |\
    tr '' '\n'
```

Application of this code to the original function definition (2) leads to the code in (1) which can be processed using the beforementioned C macros.

The extension of this simple but limited<sup>3</sup> method to arbitrary code is not the subject of this short article.

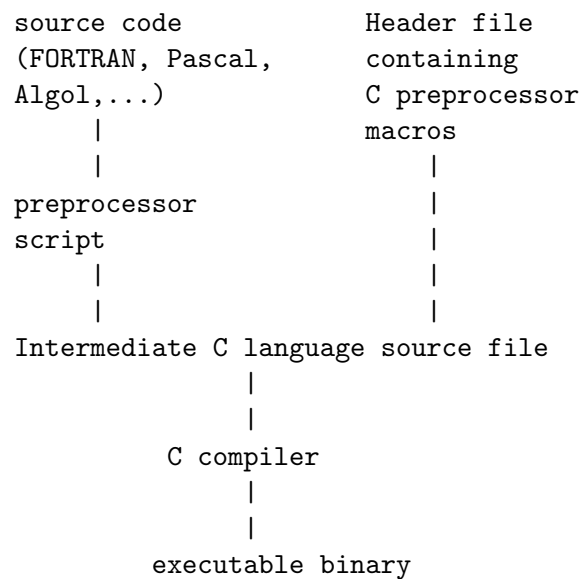
<sup>2</sup>For simplicity's sake and for the purpose of this example, we assume that file `function.pas` contains one function definition only.

<sup>3</sup>Depending on the complexity of the original function definition, the necessary preprocessor script will become too involved to be presented in this short article.

## 5 General Code Transformation Scheme

Generally speaking, each token or construction that we are not able to represent or evaluate using a C preprocessor macro should first be transformed to intermediate code that can then be evaluated by a C compiler.

**General transformation scheme for making code from FORTRAN, Pascal, or other languages compilable with a C compiler:**



This could sometimes be a viable workaround if a compiler generator definition for `yacc`, or `bison`, is unavailable or if there is only a limited number of sources available which would make purchasing or generating a compiler ineffective.

## 6 Summary

The beforementioned selection of examples exemplified the possibility of transforming code fragments written in FORTRAN, Pascal or other languages to C using, in succession, suitable preprocessor scripts and C macro definitions. Due to the complexity of the preprocessor scripts needed, this approach of circumventing the use of a compiler for these languages is not recommended in general but only in those situations where the design and coding of the necessary scripts and macros is cheaper than purchasing a new compiler.

CONTINUED IN ONE OF THE NEXT EDITIONS

RD-Avenue<sup>®</sup> Lab.Note 2011-1:  
Elementary Remarks on Using  
Interrupts in Real-time Systems

© 2010–2011 by Rudolf R.H. Dittrich  
D-82024 Taufkirchen, Germany

THE FOLLOWING recommendations for securing a system using interrupts have been found by the author to occasionally help in creating a system that is working in a stable manner. Being of an ad-hoc and rule-of-thumb nature, these recommendations should always be followed by a thorough theoretical analysis of the system before being used in a customer project:

- Beware of hardware-induced problems which could cause spurious interrupts; introduce hardware and software filters like, for example, suitable debouncing mechanisms for suppressing unwanted signal noise; if possible, or appropriate, use ADC or digital input data for verifying the cause or plausibility of an interrupt.
- Interrupt functions should be prevented from being invoked too often; access to non-reentrant functions should always be blocked if the function is still being evaluated.
- Prioritize interrupts to make sure that execution of code for processing important data is not being delayed by less important interrupts. Mask interrupts if necessary.
- Timer interrupt functions should not normally be interrupted. Consider carefully the possibly detrimental consequences of lowering a timer interrupt's priority (depends on the processor and the operating system used)
- Using a timer interrupt for data acquisition tasks: does the clock rate respect the sampling theorem's minimum requirements?
- While debugging, it may come in handy to have every unused interrupt be mapped in the vector table to dummy interrupt debug functions. In production code, these debug functions are not usually needed.
- Use resource locking mechanisms and other measures to prevent problems arising if different interrupts or operating system tasks would simultaneously access the same data — be it on the heap, the stack, an external storage device; etc.
- Make sure interrupts are not causing regular tasks<sup>1</sup> to be interrupted for too long a time. This almost always calls for a thorough analysis of the operating system used. If possible, the amount of data processed by an interrupt function and the number of interrupts used should be kept to a minimum.

It should always be borne in mind that these and similar recommendations constitute but a tiny fraction of the points to be considered while designing a real-time system.

CONTINUED IN ONE OF THE NEXT EDITIONS...

---

<sup>1</sup>We assume that either a real-time operating system or a simple time-slicing mechanism be used.

RD-Avenue<sup>®</sup> Lab.Note 2011-2

## Mailbox File Generation

© 2010–2011 by Rudolf R.H. Dittrich  
D-82024 Taufkirchen, Germany**Abstract**

The article presents a non-sophisticated shell script designed for assembling simple mailbox files from arbitrary text files which may subsequently be imported by a variety of mail programs for future processing. The script thus constitutes a simple means for adding non-mail related material to one's email system without any need on part of the user to manually select the file and send an email to his or her own address.

## 1 Introduction

OCCASIONALLY wishing to add notes and other material to his mail archive, the author wrote a simple shell script for assembling a Unix-like mailbox file containing an arbitrary ANSI-encoded text file which is thus capable of being imported by most email clients. The author does not believe that the approach of assembling a mailbox file using a shell script is unique. Substituting a high-level language

for the shell script would have resulted in a much faster and more versatile solution which could more easily be extended to include binary files like, for example, images or compiled program code. Due to its simplicity, the following shell script approach is nonetheless presented as a starting point for future mailbox experiments.

## 2 Structure of Mailbox Files

Mailbox files usually consist of a header section which is being followed by a contents section with both parts being separated by an empty line. In the following simplified example which suffices for our needs, the header section contains at least<sup>1</sup>

- a "from" line indicating the email address of the sender; and the date and time the mail was received:

```
From mailSender@sendingHost Su Jun 20 06:00:00 pm 2010
```

- a line starting with "Received:" indicating the machine receiving the email, the senders email address, information on the mail portal used and, separated by a semi-colon, the date of reception:

```
Received: from mailClient by mailSender@sendingHost with SMTP \
(127.0.0.1) id 1234567; Su Jun 20 06:00:00 pm 2010
```

<sup>1</sup>In this section's example, we assume the mail to be sent Su Jun 20, 2010 at 18:00:00.

- a return path line:

```
Return-Path: <mailSender@sendingHost>
```

- a line indicating the date and time the mail was sent:

```
Date: Su Jun 20 06:00:00 pm 2010
```

- a line, starting with "To:", which contains the recipient's email address:

```
To: "No Name" <NoName@noHost>
```

- a subject line, starting with the keyword "Subject", followed by a colon:

```
Subject: This is an example subject line
```

The text section following the header may contain arbitrary 7-bit ANSI encoded text.

Real mailbox files may contain a variety of additional pieces of information which are not needed for our mail generation script.

The beformentioned file structure can easily be generated by a small Bourne shell script, see the following section depicting the script's program code.

### 3 Program Code

The following lines contain a print-out of the mailbox generator script called `mkmailnote.sh`:

```
#!/bin/sh
# 08/20/2009: New script for putting the contents of text message files into
#           mail box files
# 11/28/2010: Modified wording of domain and user names
if [ "$#" -lt 9 ]; then
    echo "ERROR: INSUFFICIENT NUMBER OF ARGUMENTS" 1>&2
    echo "USAGE:" 1>&2
    echo "mkmailnote.sh DATE TIME GATEWAYIDENTIFIER FROMNAME TONAME TOADDR \"\
    \"SUBJECT \"\" 1>&2
    echo "   SRCFILE TARGETFILE" 1>&2
    echo "EXAMPLE:" 1>&2
    echo "mkmailnote.sh 1998-01-01 12:00:00 note \"NameOfSender\" \"\
    \"\"NameOfRecipient\" \"\" 1>&2
    echo "   recipient@toDomain \"\" 1>&2
    echo "   \"New note subject line\" \"\" 1>&2
    echo "   note.txt note.mail" 1>&2
    exit 1
else
    GATEWAYADDR=127.0.0.1
    MYDOMAIN=mydomain
    LDATE=$1
```

```

LTIME=$2
LOGGATEWAYNAME=${3}gateway@localhost
LOCFROMADDRESS="\${4}\" <${3}gateway@dummys${3}gateway.${MYDOMAIN}>"
LOCTONAME="\${5}\""
LOCTOADDR="<${6}>"
LOCSUBJECT="$7"
LOCTXTFILE="$8"
LOCTARGFILE="$9"
LASTID='tail -1 $HOME/.mkmailids'
LOCID='expr $LASTID + 1'
LOCDATE='LOCALE=C LANG=en date +%a %b %d %I:%M:%S %P %Y' -d"$LDATE $LTIME"
echo "From $LOGGATEWAYNAME $LOCDATE" >> $LOCTARGFILE
echo "Received: from localhost by $LOGGATEWAYNAME with SMTP ($GATEWAYADDR)" \
" id AA$LOCID; $LOCDATE" >> $LOCTARGFILE
echo "Return-Path: <${LOGGATEWAYNAME}>" >> $LOCTARGFILE
echo "Date: $LOCDATE" >> $LOCTARGFILE
echo "From: $LOCFROMADDRESS" >> $LOCTARGFILE
echo "To: $LOCTONAME $LOCTOADDR" >> $LOCTARGFILE
echo "Subject: $LOCSUBJECT" >> $LOCTARGFILE
echo "" >> $LOCTARGFILE
cat "$LOCTXTFILE" >> $LOCTARGFILE
echo >> $LOCTARGFILE
echo "$LOCID" >> $HOME/.mkmailids
fi

```

## 4 Example

For testing purposes, we may invoke the mailbox script using the following parameters:

```

./mkmailnote.sh 1998-01-01 12:01:02 note "My Name" "Recipients Name" \
recipient@nowhere "New test note" tst101128.txt tst101128.mail

```

where `tst101128.txt` is a text file containing the following lines:

```

Test line 1
test line 2
last test line

```

In this example, we additionally assume that the last line of file `$HOME/.mkmailids` consists of the following number which will be incremented by the script to obtain a unique mail id:

```

55004

```

## 5 Resulting Mailbox File

The structure of the resulting mailbox file corresponds to the contents of other mailbox files as detailed in section 2. It may therefore be imported by a variety of email programs:

```
From notegateway@localhost Thu Jan 01 12:01:02 pm 1998
Received: from localhost by notegateway@localhost with SMTP (127.0.0.1) \
id AA55005; Thu Jan 01 12:01:02 pm 1998
Return-Path: <notegateway@localhost>
Date: Thu Jan 01 12:01:02 pm 1998
From: "My Name" <notegateway@dummynotegateway.mydomain>
To: "Recipients Name" <recipient@nowhere>
Subject: New test note
```

```
Test line 1
test line 2
last test line
```

## 6 Summary

The simple script presented in this article constitutes just one way of writing a tool for copying the contents of an arbitrary ANSI-encoded text file into a mailing system — without any need on part of the user to actually write an email. The beforementioned script was designed for text files only. The extensions necessary for importing image or other binary files, including non-ANSI encoded texts, are left as an exercise for the interested reader.

RD-Avenue<sup>®</sup> Lab.Note 2011-3

### Multi-line Text Searching and the `sed` Command

© 2010–2011 by Rudolf R.H. Dittrich  
D-82024 Taufkirchen, Germany

Quite often, we encounter the prejudicial statement that single line-oriented text manipulation tools like the `sed` command cannot be used if a search pattern would span more than a single line. Advice is then usually given which recommends using languages like `Perl` or other tools that are specifically designed to include multi-line search patterns.

For many years, the author has found no particular need to heed such advice but was instead using the following simple method for implementing multi-line search patterns in conjunction with `sed`:

Upon replacing a file's newline characters, i.e. `'\n'`, by a temporary replacement string or, in case the text is not using all possible characters, by a single character, it is possible to replace all lines of the file by a single line.

This line may then be processed by the `sed` command.

Finally, we may replace again the temporary string, or character, by the original newline `'\n'`.

By way of example, let us assume that the % character is not part of our file. The beforementioned multi-line search algorithm can then be used in a script as follows:

```
cat file.txt |\
tr '\n' '%' |\
sed 's/multilineSearchPattern/ReplacementString/g' |\
tr '%' '\n'
```

The same procedure may be followed while using the `grep` command or any other line-oriented program.  
CONTINUED IN NEXT YEAR'S EDITION...

## Auxiliary Tools for Visually Checking Verbs Followed by the Gerund or the Infinitive

© 2010–2011 by Rudolf R.H. Dittrich  
D-82024 Taufkirchen, Germany

### Abstract

The present article is neither intending to present a general introduction to English grammar nor does it contain a comprehensive or in any other way exhaustive overview describing how to decide whether a verb should be followed by an infinitive construction, or the gerund. We present two simple auxiliary text tools for finding and visually highlighting a set of verbs for which use of the gerund or the infinitive is well-documented in the literature. Those interested in linguistics may extend the tools if deemed necessary. The non-sophisticated text search algorithm used in both tools implies, however, that each search result should always be supplemented by a careful visual inspection of the text.

## 1 Introduction

REWRITING an article or changing the wording of a text may occasionally introduce errors if not being done carefully. One typical class of problems concerning the usage of verbs is often encountered when using a text editor program's automatic replacement feature for exchanging a verb by another.

This may occasionally cause errors in style or grammar. By way of example, let us consider the automatic replacement of the verb *to refuse* by *to dislike*:  
A sentence like

*he refused to drink coffee*

would then be transformed into

*he disliked to drink coffee*

*To dislike* belonging to a group of verbs usually being followed by the gerund<sup>1</sup>, this should better read

*he disliked drinking coffee*

The present article does not claim to offer an easy solution to such grammar related text processing issues. Instead we present a short auxiliary Bash script for addressing part of this problem. The script, which should run under most flavours of the Linux or Unix operating systems, searches text passages for a selection of verbs which the user may then rewrite if deemed necessary.

We will not consider verbs for which no preferential rule for adding the gerund or the infinitive exists. Such verbs would include

begin  
continue  
hate  
intend  
like  
love  
prefer  
start

etc. Other examples may be looked up in the literature.

## 2 Examples of Verbs Usually Followed by the Gerund

The following examples further highlight the necessity to check whether replacing a verb should call for the gerund. In all of the following sentences, we are using the infinitive after the verb:

1. He neglected to solder the wire.
2. The engineering department promised to spend the money.
3. The executive board failed to buy the premise.

Upon replacing *neglect* by *resist*, *promise* by *stop*, and *fail* by *contemplate*, the infinitive should be replaced by the gerund as follows:

1. He resisted soldering the wire.
2. The engineering department stopped spending the money.
3. The executive board contemplated buying the premise.

The following list of verbs, usually calling for the gerund, is far from exhaustive:

|            |             |          |
|------------|-------------|----------|
| admit      | can't help  | defer    |
| adore      | carry on    | delay    |
| advise     | complete    | deny     |
| anticipate | consider    | describe |
| appreciate | contemplate | detest   |
| avoid      | describe    |          |

<sup>1</sup>Note that we speak of verbs being followed by another verb. This does not imply in any way that use of a second verb following the first one is mandatory. Depending on the context, almost all of these verbs may be followed by an object without any need to introduce a verb.

|               |            |              |
|---------------|------------|--------------|
| discuss       | involve    | report       |
| dislike       | justify    | resent       |
| end up        | keep on    | reserve      |
| enjoy         | listen to  | resist       |
| entail        | look after | resume       |
| escape        | mention    | risk         |
| excuse        | mind       | see          |
| fancy         | miss       | sense        |
| feel          | notice     | sleep        |
| finish        | observe    | spend        |
| have finished | perceive   | stop         |
| forbid        | permit     | suggest      |
| get through   | plan on    | take care of |
| give up       | postpone   | tolerate     |
| go on         | practice   | try          |
| hear          | quit       | understand   |
| imagine       | recall     | waste        |
| include       | recommend  | watch        |
| insist on     | regret     |              |

### 3 An Auxiliary Tool for Displaying Verbs Usually Followed by the Gerund

In this section we present "verbs4gerundFinder.sh", a rather simple auxiliary script for highlighting the verbs presented in the above list. Not being sophisticated enough to distinguish between verbs and other word classes, or their usage, the script's output of the `grep` command is meant to be used for visually inspecting text passages that might contain errors in grammar. Due to limitations of the `grep` command which does not have any knowledge of tenses, or the semantics of a sentence including the distinction between verbs and nouns, the beforementioned list was slightly augmented by adding the 3rd person singular of the present tense, the preterite and, if differing from the preterite, the verb's past participle, and the gerund. Please note that the script currently contains a limited number of verbs only and should be substantially enlarged before being used for checking arbitrarily chosen texts. Invocation of the script is explained in section 5.

```
#!/bin/bash
# Script verbs4gerundFinder.sh
WAITFORPROMPT=y
if [ "$1" = "-w" ]; then
    WAITFORPROMPT=n
    shift
fi
```

for word in 'accuse of' 'accuses of' 'accused of' 'accusing of' \  
 admit admits admitted admitting \  
 adore adores adored adoring \  
 advise advises advised advising \  
 anticipate anticipates anticipated anticipating \  
 appreciate appreciates appreciated appreciating \  
 avoid avoids avoided avoiding \  
 "can't help" 'cannot help' 'can not help' \  
 "couldn't help" 'could not help' \  
 'carry on' 'carries on' 'carried on' 'carrying on' \  
 complete completes completed completing \  
 consider considers considered considering \  
 contemplate contemplates contemplated contemplating \  
 defer defers deferred deferring \  
 delay delays delayed delaying \  
 deny denies denied denying \  
 describe describes described describing \  
 detest detests detested detesting \  
 discuss discusses discussed discussing \  
 dislike dislikes disliked disliking \  
 'end up' 'ends up' 'ended up' 'ending up' \  
 enjoy enjoys enjoyed enjoying \  
 entail entails entailed entailing \  
 escape escapes escaped escaping \  
 excuse excuses excused excusing \  
 fancy fancies fancied fancying \  
 feel feels felt feeling \  
 finish finishes finished finishing \  
 'have finished' 'has finished' 'had finished' 'having finished' \  
 forbid forbids forbade forbidding \  
 'get through' 'gets through' 'got through' 'gotten through' \  
 'getting through' \  
 'give up' 'gives up' 'gave up' 'given up' 'giving up' \  
 'go on' 'goes on' 'went on' 'going on' \  
 hear hears heard hearing \  
 imagine imagines imagined imagining \  
 include includes included including \  
 'insist on' 'insists on' 'insisted on' 'insisting on' \  
 involve involves involved involving \  
 justify justifies justified justifying \  
 'keep on' 'keeps on' 'kept on' 'keeping on' \  
 keep keeps kept keeping \

```

'listen to' 'listens to' 'listened to' 'listening to' \
'look after' 'looks after' 'looked after' 'looking after' \
mention mentions mentioned mentioning \
mind minds minded minding \
miss misses missed missing \
notice notices noticed noticing \
observe observes observed observing \
perceive perceives perceived perceiving \
permit permits permitted permitting \
'plan on' 'plans on' 'planned on' 'planning on' \
postpone postpones postponed postponing \
practice practices practiced practicing \
quit quits quitted quitting \
recall recalls recalled recalling \
recommend recommends recommended recommending \
regret regrets regretted regretting \
report reports reported reporting \
resent resents resented resenting \
reserve reserves reserved reserving \
resist resists resisted resisting \
resume resumes resumed resuming \
risk risks risked risking \
see sees saw seen seeing \
sense senses sensed sensing \
sleep sleeps slept sleeping \
spend spends spent spending \
stop stops stopped stopping \
suggest suggests suggested suggesting \
'take care of' 'takes care of' 'took care of' \
'taken care of' 'taking care of' \
tolerate tolerates tolerated tolerating \
try tries tried trying \
understand understands understood understanding \
waste wastes wasted wasting \
watch watches watched watching; do
w='echo -n $word|sed 's/ / \.* /g','
erg="'grep -w "$w" $*'"
if [ "$erg" != "" ]; then
    echo
    if [ "$word" = "spend" -o "$word" = "spent" -o "$word" = "waste" ]; \
    then
        echo "----- GERUND CHECK $word (time ONLY)"

```

```

else
    echo "----- GERUND CHECK $word"
fi
grep --color -w "$w" $*
if [ "$WAITFORPROMPT" = "y" ]; then
    echo -n "PRESS RETURN TO CONTINUE> "
    read answerOnPrompt
fi
fi
done

```

## 4 Auxiliary Shell Script for Checking Verbs Followed by the Infinitive

In the preceding sections, we focused on verbs followed by the gerund. We also mentioned verbs that may be followed by the gerund or the infinitive. Some verbs, however, usually call for the infinitive only. The following additional shell script, called "verbs4infinitiveFinder.sh", scans an arbitrary English language text<sup>2</sup> searching for and, if found, displaying a subset of these verbs. The script is being invoked in exactly the same way as "verbs4gerundFinder.sh", see section 5 for additional detail.

```

#!/bin/bash
# Script verbs4infinitiveFinder.sh
WAITFORPROMPT=y
if [ "$1" = "-w" ]; then
    WAITFORPROMPT=n
    shift
fi
for word in afford affords afforded affording \
agree agrees agreed agreeing \
appear appears appeared appearing \
arrange arranges arranged arranging \
beg begs begged begging \
care cares cared caring \
claim claims claimed claiming \
consent consents consented consenting \
decide decides decided deciding \
demand demands demanded demanding \
deserve deserves deserved deserving \
expect expects expected expecting \
fail fails failed failing \
hesitate hesitates hesitated hesitating \

```

<sup>2</sup>We consider ANSI-encoded text files only.

```

hope hopes hoped hoping \
learn learns learned learnt learning \
manage manages managed managing \
neglect neglects neglected neglecting \
need needs needed needing \
offer offers offered offering \
plan plans planned planning \
prepare prepares prepared preparing \
pretend pretends pretended pretending \
promise promises promised promising \
refuse refuses refused refusing \
seem seems seemed seeming \
struggle struggles struggled struggling \
swear swears swore sworn swearing \
threaten threatens threatened threatening \
volunteer volunteers volunteered volunteering \
wait waits waited waiting \
want wants wanted wanting \
wish wishes wished wishing; do
erg="'cat $*|tr '\n' ' '|sed 's/\./\.\n/g'|grep -w "$word"'
if [ "$erg" != "" ]; then
    echo
    echo "----- INFINITIVE CHECK $word"
    cat $*|tr '\n' ' '|sed 's/\./\.\n/g'|grep --color -w "$word"
    if [ "$WAITFORPROMPT" = "y" ]; then
        echo -n "PRESS RETURN TO CONTINUE> "
        read answerOnPrompt
    fi
fi
done

```

## 5 Invocation of Tools

The Bash scripts introduced in the preceding sections may be used on ANSI encoded text files as follows.

### 5.1 Invocation of `verbs4gerundFinder.sh`

The first script may be invoked by issuing the following command:

```
verbs4gerundFinder.sh textfile.txt
```

where `textfile.txt` is the name of the text file. Issuing the slightly modified command

```
verbs4gerundFinder.sh -w textfile.txt
```

prevents the user from having to press the RETURN key after each search result. Specifying more than one text file causes the search to extend to each of these files:

```
verbs4gerundFinder.sh file1.txt file2.txt file3.txt
```

## 5.2 Invocation of verbs4infinitiveFinder.sh

Using the same parameters, invocation of "verbs4infinitiveFinder.sh corresponds exactly to the first script:

```
verbs4infinitiveFinder.sh textfile.txt
```

or

```
verbs4infinitiveFinder.sh -w file1.txt file2.txt file3.txt
```

etc.

## 6 Summary

The simple scripts presented in this article display text passages containing a predefined set of search phrases and may thus be used for visually inspecting ANSI-encoded text files in order to assist the user in deciding whether verbs followed by an infinitive or gerund construction have been used in a grammatically proper way. Due to the absence of any program code deciding on the semantic correctness of a text, the usefulness of the scripts is limited, consisting mainly in the visual highlighting of the search results. Although lacking any means for automatically deciding on the replacement of the infinitive by the gerund, or vice versa, the author found the scripts to be occasionally useful for finding errors in grammar that would otherwise have remained undetected.

UPCOMING IN ONE OF THE NEXT ISSUES:  
REMARKS ON USING ADA, PART II  
UPDATING A VERY OLD LINUX SYSTEM, PART II  
FURTHER REMARKS ON LINUX PACKAGE ADMINISTRATION  
ENGINEERING TRANSLATIONS: SOME INTRODUCTORY EXAMPLES  
AN ELEMENTARY INTRODUCTION TO GENERAL RELATIVITY  
THE AUTHOR'S SHORT INTRODUCTION TO L<sup>A</sup>T<sub>E</sub>X, Part II

**At the time of this writing, the following link was provided for downloading this issue<sup>3</sup>:**  
[http://www.rudolf-rh-dittrich.com/img/rudolf-rh-dittrich.com\\_  
RudolfRHDittrich\\_InternetPublicationSeries\\_VOL\\_2010\\_2011\\_3rd\\_printing\\_LetterSize.pdf](http://www.rudolf-rh-dittrich.com/img/rudolf-rh-dittrich.com_RudolfRHDittrich_InternetPublicationSeries_VOL_2010_2011_3rd_printing_LetterSize.pdf)

---

<sup>3</sup>Link may be unavailable at the time this page is being read.

Note that above link was split into two lines for printing purposes only; both lines should be concatenated to obtain original link.