Sample Application: Frequency of words in text

It is easy to install `cygwin` on an `XP` machine to get the `bash` command-line interface. In addition, `cmd.exe` runs natively on `XP`, and `powershell` v1.0 was released to try out on `XP` in November 2006.

Let us suppose that you boot to your GUI (Graphical User Interface) and that you want to perform this very simple task:

```
display the most frequent words in any given text document
```

We search the GUI menus — but no luck; we search the supplied commands — but still no luck. So we have to do this ourselves, by developing a command script of our own.

Develop Script: Frequency of words in text

Here I trace the steps you might use in developing a script for this task.

Original text in the file:

```
This is text in the file       TRANSLATE      this is text in the file
with numbers (85) & symbols, UPPER TO LOWER   with numbers (85) & symbols,
[like this:], OK?                             [like this:], ok?

this is text in the file        CONVERT       this is text in the file
with numbers (85) & symbols,  NON_LETTERS     with numbers        symbols
[like this:], ok?                              like this    ok

this is text in the file        REMOVE        this is text in the file
with numbers      symbols    MULTIPLE SPACES  with numbers symbols
 like this    ok                              like this ok

1/LINE -----> SORTED ----> COUNTED --> REVERSE NUMBER SORT -> TOP 5

this          file         1 file           2 in              2 in
is            in           1 in             2 this            2 this
text          is           1 is             1 file            1 file
in            in           2 in             1 like            1 like
the           like         1 like           1 numbers         1 numbers
file          numbers      1 numbers        1 ok
with          ok           1 ok             1 symbols
numbers       symbols      1 symbols        1 the
symbols       the          1 the            1 with
like          this         2 this
this          this         1 with
ok            with
```

CMD: Frequency of words in text

First, install the stream-editor `sed`. You may do this by going to `http://garbo.uwasa.fi/gsearch.html` and entering `sed` in the search bar; choose any version of `sed` for XP that appeals to you.

We start with a *main routine* called `freq.bat` that invokes `separate.bat` (which separates words into lines) and `distinct.bat` (which counts distinct words).

```
create script 'freq.bat':     cmd> notepad freq.bat  [and type in the text below]
create environment:           echo off & setlocal
allocate parameters:          set number=%1 & set FILE=%2
                              if %number% LEQ 0 set /a number=10
check text file exists:       :validate
                              if NOT exist %FILE% (
                                set /p FILE="enter filename< "
                                goto :validate )
upper case to lower:          type %FILE%|^
next 2 lines on one line:     sed "y/ABCDEFHGIJKLMNOPQRSTUVWXYZ"
                                   /abcdefghijklmnopqrstuvwxyz/"|^
non-letters to spaces:        sed "s/[^a-z]/ /g"|^
ignore any empty lines:       sed "/^$/d"|^
save the translated text:     >trans.txt
separate words on own line:   call separate.bat
count distinct words:         call distinct.bat
select the most frequent:     sed -n 1,%number%p distinct.txt
cancel local variables:       endlocal
all done:                     exit /b 0
(end of freq.bat script):     ^S      [CONTROL S saves the file]
```

```
create script 'separate.bat':
        cmd> notepad  separate.bat   [and type in the following text]
        :: separate.bat
        echo off & setlocal & if exist words.txt del words.txt
        :: list all words in each line of file trans.txt
        for /f "tokens=* delims=" %%a in ('type trans.txt') do call :extract %%a
        :: sort the words in alphabetic order
        type words.txt | sort > separate.txt
        endlocal & goto :eof
        :: routine to extract words and output one word per line
        :extract
        if [%1]==[] goto :eof
        echo.%1>>words.txt
        shift & goto :extract
        ^S        [CONTROL S saves the script]
```

```
create script distinct.bat:
                    cmd> notepad distinct.bat   [and type in the following text]
                    echo off
                    setlocal enableextensions enabledelayedexpansion
                    if exist counted.txt del counted.txt
                    set WORD=######
                    set /a COUNT=0
                    for /f "tokens=1 delims=" %%w in ('type separate.txt') do (
                    if "%%w" == "!WORD!"  set /a COUNT+=1
                    if not "%%w" == "!WORD!" call :format !COUNT! !WORD!
                    if not "%%w" == "!WORD!" set /a COUNT=0
                    if not "%%w" == "!WORD!" set WORD=%%w
                    )
                    type counted.txt|find /v "######"|sort /r>distinct.txt
                    endlocal
                    goto :eof
                    :: routine to format numbers so they sort numerically
                    :format
                    if %1 LSS 10     echo.    %1 %2>>counted.txt & goto :eof
                    if %1 LSS 100    echo.   %1 %2>>counted.txt & goto :eof
                    if %1 LSS 1000   echo.  %1 %2>>counted.txt & goto :eof
                    if %1 LSS 10000  echo. %1 %2>>counted.txt & goto :eof
                    if %1 LSS 100000 echo.%1 %2>>counted.txt & goto :eof
                    goto :eof
                    ^S        [CONTROL S to save the script]
locate the "freq" script:   cmd> doskey freq="%HOMEPATH%\freq.bat" $*
now to invoke the script:   cmd> freq 15 some.txt
```

PSH: Frequency of words in text

```
(this symbol ` is a backtick, under the ~ (tilde) key, for continuing lines)
allow execution of scripts:     psh> Set-ExecutionPolicy RemoteSigned
create script 'freq.ps1':       psh>   cd; notepad freq.ps1
allocate parameters:            $number=[int]$args[0] ; $textfile=[string]$args[1]
check file existence:           while (!(test-path ./$textfile -pathtype leaf)) {
repeat until valid:               $textfile=read-host -prompt "enter filename< "}
put text into single string:    $singleline=get-content $textfile
turn it into one long line:     $longline=[string]::join(" ",$singleline)
convert upper to lower case:    $lowercase=$longline.ToLower()
non-letters to spaces:          $letters=$lowercase -replace '[^a-z]',' '
each word onto own line:        $words=$letters.split(" ",[StringSplitOptions]::
(all this on one line):           RemoveEmptyEntries)
```

```
take all the words so formed    $words ‘
and group by same word:           | group-object ‘
and sort in occurrence order:     | sort -descending count ‘
print out what we want:           | format-table -property count,name ‘
without table headings:              -hidetableheaders ‘
and see most frequent:            | select-object -first $number
all done:                       exit 0
[for a faster hashtable method, replace "$words...exit 0" above with this gibberish*]
*take all the words so formed:  $words ‘
*and group by same word:          | % {$h=@{}} {$h[$_]+=1}
*and sort in occurrence order:  $frequency=$h.psbase.keys|sort {$h[$_]}
*print out the most frequent:   -1..-$number|%{$frequency[$_]+" "+$h[$frequency[$_]]}
*all done:                      exit 0
(end of freq.ps1 script):       ^S      [this is CONTROL S] then close window
locate the ’freq’ script:       > function freq($N,$FILE) {
                                  & "$HOME\freq.ps1" $N $FILE}
now can run the program:        > freq 15 some.txt
```

BASH: Frequency of words in text

Our main script is `freq`, which is where we carry out our sanity tests first.

```
create script ’freq’:           bash>     cat>freq
run script with bash:           #!/bin/bash
allocate parameters:            number=$1; textfile=$2
check existence of file:        while [ ! -e "$textfile" ]; do
request missing filename:       read -p "enter text filename < " textfile; done
number defaults to 10:          if [ "$number" -lt 0 ]; then let number=10; fi
get the text file:              cat $textfile \
upper case to lower:            |tr ’[A-Z]’ ’[a-z]’ \
non-letters to spaces:          |sed ’s/[^a-z]/ /g’ \
squeeze multiple spaces:        |tr -s ’ ’ \
words onto single lines:        |tr ’ ’ ’\n’ \
exclude any empty lines:        |grep -v ^$ \
sort words alphabetically:      |sort \
count number of each word:      |uniq -c \
sort in occurrence order:       |sort -n -r \
show the most frequent:         |head -$number
all done:                       exit 0
(finish of freq script):        ^D   [CONTROL D ends input, then press ENTER]
locate the ’freq’ script:       bash> alias freq=’$HOME/freq’
and run the script:             bash> chmod +x freq; freq 15 some.txt
```

Execution Timing Results

The time taken for this task, under XP and all on the same 1.6GHz ACER laptop, was:

Shell	Text Size	Time Taken	Comments and Notes
cmd	118 Kb	75 seconds	slow due to awkward unsuitable design
	1180 Kb	859 seconds	
	11800 Kb		not attempted
psh	118 Kb	19 seconds	slow because negotiating .NET, objects, &c.
	118 Kb	9 seconds	using advanced but more cryptic hashtable method
	11800 Kb	254 seconds	using advanced but more cryptic hashtable method
bash	118 Kb	0.75 seconds	open-source code gets comprehensively optimised
	11800 Kb	38 seconds	

Comparison of Shells

parameter specification is very similar in all shells.

```
cmd>  %1 %2           [but you need to use %%1 %%2 in a batch file]
psh>  $1 $2 ... $9    [but there's no 'shift' command! - what were they thinking?]
bash>  $1 $2 ... $9   [and you use 'shift' to feed in more variables > 9]
```

variable allocation has some tricks up its sleeve.

```
cmd>   set n="%1"   [generally]        set /a n=%1    [for numbers]
psh>   $n="$1"      [generally]        $n=[int]$1     [for numbers]
bash>  n="$1"       [generally]        let n=$1       [for numbers]
```

referring to variables is very similar in all shells.

```
cmd>   echo %filename%
psh>   out-host "$filename"
bash>  echo "$filename"
```

continuing command lines is very similar in all these shells.

```
cmd>   type file.txt | ^          [the circumflex ^ continues lines]
psh>   get-content file.txt | `   [the backtick ` continues lines]
bash>  cat file.txt | \           [the backslash \ continues lines]
```

conditional execution has some little wrinkles.

```
cmd>   if "%option%" == "test"   set /a x=3          [the /a is for arithmetic]
psh>   if ( "$option" -eq "test" ) { $x=3 }          [more like C# program syntax]
bash>  if [ "$option" == "test" ]; then let x=3; fi  [if-then-fi delimits action]
```

executing a chain of commands is very similar in all these shells.

```
cmd>   now & freq 15 big.txt & now      [the & separates subsequent commands]
psh>   now; freq 15 big.txt; now        [the ; separates subsequent commands]
bash>  date; freq 15 big.txt; date      [the ; separates subsequent commands]
```

defining aliases is quite different in each shell.

```
cmd>   doskey freq="%HOMEPATH%\freq.bat" $*      [$* means all script parameters]
psh>   function freq($n,$text) { & "$HOME\freq.ps1" $n $text }  [note parameters]
bash>  alias freq='$HOME/freq'                   [requires no parameter spec.]
```

control loops have some slight differences in format.

```
cmd>   for %a in (%LIST%) do (
         call script.bat %a  )                    [note 'call' to invoke script]
psh>   foreach ($a in $LIST) {
         ./script.ps1 $a     }                    [note $a PERL heritage]
bash>  for f in $LIST; do ./script $f; done       [do-done delimits the action]
```

But watch out for these ...

multi-line command editting is very different. I consider it completely broken in `cmd` and `psh`.

```
cmd>   for %a in (%LIST%) do (
More?    call script.bat %a
More?  )                              [now press the UP ARROW to edit this]
cmd>   )                  [which is useless; it's only the last character typed]

psh>   foreach ($a) in ($LIST) {
>>       ./script.ps1 $a
>>     }                              [now press the UP ARROW to edit this]
psh>   }                  [which is useless; it's only the last character typed]

bash>  for f in $LIST
>      do
>        ./script $f
>      done                           [now press the UP ARROW to edit this]
bash>  for f in $LIST; do ./script $f; done     [the whole command ready to edit]
```

when referring to multiple items be careful with the unusual comma-delimited list.

```
cmd>  del a b c                       [works fine]
cmd>  type a b c>x                    [works fine]
```

```
psh>  remove-item a,b,c              [you need commas, not spaces]
psh>  cat a,b,c >x                   [you need the space in 'c >']
bash> rm a b c                       [works fine]
bash> cat a b c>x                    [works fine]
```

getting user input from the terminal has some idiosyncracies.

```
cmd>   set /p file="enter filename< "              [gets a file name from user]
psh>   $file=read-host -prompt "enter filename< "  [gets a file name from user]
bash>  read -p "enter filename < " file            [gets a file name from user]
```

paging a file within a script in `psh` fails to return when you quit using `out-host -paging` or `more`; both are completely broken — you need the older `C:\WINDOWS\system32\more` instead!

```
psh>  get-content $file|C:\windows\system32\more    NOT: get-content $file|more
```

timing a Command: if you want to time how long a script takes, `bash` does that *and* produces the normal result of the script, but `psh` fails to produce the result of running the script at all!

```
cmd>  now & script.bat & now         [works fine, but calculate time yourself]
psh>  measure-command {./script}     [produces NO OUTPUT from the script]
bash> time ./script                  [normal output plus execution time]
```