Eksamensinformasjon

Bokmål

Eksamensoppgave

Operativsystemer (DATA2500)

Bokmål

5. juni 2018
kl. 9.00-12.00

Hjelpemidler:
Ingen hjelpemidler er tillatt.

Andre opplysninger:

Les nøye gjennom oppgavene før du begynner og pass på å besvare alle spørsmålene.
Sett gjerne egne forutsetninger dersom du synes oppgaveteksten er uklar. Beskriv forutsetningene og løs oppgaven utifra dem.

Maks poeng på de fleste av deloppgavene er 10 og de bidrar 3.33% hver til sluttpoengsummen som maksimalt er 300. Men legg merke til at deloppgavene 2 og 5(c) teller mer. Oppgave 2 teller 20% (60 poeng) og oppgåve 5(c) tellller 10%(30 poeng).

1(a) Linux kommandolinje

Hva blir i et standard bash-shell resultatet av kommandoen

$ **move /bin/rm /usr/bin**

Velg ett alternativ

- move: command not found
- move: cannot move '/bin/rm' to '/usr/bin/rm': Permission denied
- move: cannot open '/bin/rm' for reading: Permission denied
- Filen /bin/rm flyttes til katalogen /usr/bin
- Filen /bin/rm flyttes til filen /usr/bin

Maks poeng: 10

1(b) Linux kommandolinje

Hvilken kommando utfører kommandoen **hostname** på serveren os3.vlab.cs.hioa.no?

Velg ett alternativ

- scp hostname os3.vlab.cs.hioa.no
- ssh hostname os3.vlab.cs.hioa.no
- ssh os3.vlab.cs.hioa.no hostname
- ssh hostname@os3.vlab.cs.hioa.no
- scp os3.vlab.cs.hioa.no:hostname .
- wget os3.vlab.cs.hioa.no/hostname

Maks poeng: 10

1(c) Linux kommandolinje

Du har en fil **readme.txt** og utfører kommandoen
$ **cat readme.txt > /dev/null**
Hva skjer?

Velg ett alternativ

- Innholdet i readme.txt legges til innholdet i filen /dev/null
- Output fra cat omdirigeres til /dev/null og forsvinner
- Innholdet i readme.txt slettes
- En ny tom fil med navn /dev/null lages
- Innholdet i readme.txt lagres i filen /dev/null

Maks poeng: 10

1(d) Linux kommandolinje

Du har en fil **biler** med følgende innhold:
$ **cat biler**
student bmw 500000
haugerud berlingo 150000
kyrre elbil 90000

Hvilken Linux-**kommando** fører til følgende resultat?

$ **kommando**
haugerud berlingo 150000
student bmw 500000
kyrre elbil 90000

Velg ett alternativ

- sort -n biler
- sort -r -k 1 biler
- sort -n -r -k biler
- sort biler
- sort -n -k 3 biler
- sort -k 2 biler

Maks poeng: 10

1(e) Linux kommandolinje

Gitt følgende fil med navn **index.html**:

$ **cat index.html**
<!DOCTYPE html>

<html itemscope itemtype="http://schema.org/Article" xmlns="http://www.w3.org/1999/xhtml" xml:lang="no" lang="no">
<head>
<title>Hjem - HiOA</title>
<meta name="Content-Type" content="text/html; charset=utf-8" />
<meta name="Content-language" content="no-bokmaal" />

<meta name="MSSmartTagsPreventParsing" content="TRUE" />
<meta name="generator" content="eZ Publish" />
<meta name="title" content="Hjem" />
<meta itemprop="name" content="Hjem">
<meta name="author" content="" />
<meta name="p:domain_verify" content="3a11dc5033d5012ab4dce032cb87920c"/>

Hvilken kommando gir følgende output?

$ **kommando**
Content-Type
Content-language
MSSmartTagsPreventParsing
generator
title

Velg ett alternativ

- cat index.html | grep name | tail -n 5
- tail -n 5 index.html | grep name | cut -d\" -f 4
- grep name index.html | head -n 5 | cut -d\" -f 2
- cat index.html | cut -f 5 | grep name | awk '{print $2}'

Maks poeng: 10

2 Bash-scripting

I denne oppgaven skal du lage et shell-script med navn **dsh** som setter opp et lite distribuert shell som du kan bruke til å kjøre kommandoer på andre Linux-maskiner som du har ssh-tilgang til.

Brukeren som skal kjøre scriptet har en fil som heter **~/hosts** og som innholder navnet på andre Linux-hosts som eieren av scriptet har tilgang til med ssh. Denne tilgangen er satt opp med ssh-nøkler, slik at brukeren kan logge seg inn til disse Linux-maskinene uten å taste passord. Helt til slutt i oppgaveteksten vises et eksempel på hvordan **dsh** skal virke når man bruker det og i dette tilfellet innholder **~/hosts** følgende:

```
$ cat ~/hosts
nexus.cs.hioa.no
studssh.cs.hioa.no
trident2.vlab.cs.hioa.no
$
```

Når scriptet **dsh** kjøres, kan du anta at alle hosts i **~/hosts** er oppe og kan nås med ssh uten å taste passord. Det som skal skje når du starter bash-scriptet **dsh** er at du får opp et prompt **dsh$** og hvis du skriver inne en kommando til dette shellet så skal den utføres på den lokale maskinen (rex i dette eksempelet) og på alle maskiner som ligger i filen **~/hosts** og resultatet vises i terminalen som vist under. Ett unntak er hvis brukeren skriver inn kommandoen **exit** til **$dsh**, da skal shellet avsluttes og brukeren skal komme tilbake til det vanlige lokale bash-shellet. Dette er vist helt til slutt i eksempelet under.

Om du ikke husker syntaks for noen av bash-kommandoene du kan velge å bruke i dette scriptet, kan du se på denne hjelpefilen.

I eksempelet under er alle kommandoer som brukeren skriver inn vist med **uthevet** font.

```
$ ./dsh
dsh$ whoami
#### rex ####
haugerud
#### nexus.cs.hioa.no ####
haugerud
#### studssh.cs.hioa.no ####
haugerud
#### trident2.vlab.cs.hioa.no ####
haugerud
dsh$ uname -r
#### rex ####
4.4.0-119-generic
#### nexus.cs.hioa.no ####
4.4.0-93-generic
#### studssh.cs.hioa.no ####
4.4.0-101-generic
#### trident2.vlab.cs.hioa.no ####
4.4.0-116-generic
dsh$ hostname
#### rex ####
rex
#### nexus.cs.hioa.no ####
nexus
#### studssh.cs.hioa.no ####
studssh
#### trident2.vlab.cs.hioa.no ####
trident2
dsh$ php -version
#### rex ####
PHP 7.0.28-0ubuntu0.16.04.1 (cli) ( NTS )
Copyright (c) 1997-2017 The PHP Group
Zend Engine v3.0.0, Copyright (c) 1998-2017 Zend Technologies
    with Zend OPcache v7.0.28-0ubuntu0.16.04.1, Copyright (c) 1999-2017, by Zend Technologies
#### nexus.cs.hioa.no ####
PHP 7.0.22-0ubuntu0.16.04.1 (cli) ( NTS )
Copyright (c) 1997-2017 The PHP Group
Zend Engine v3.0.0, Copyright (c) 1998-2017 Zend Technologies
    with Zend OPcache v7.0.22-0ubuntu0.16.04.1, Copyright (c) 1999-2017, by Zend Technologies
#### studssh.cs.hioa.no ####
```

```
bash: php: command not found
#### trident2.vlab.cs.hioa.no ####
bash: php: command not found
dsh$ exit
Avslutter dsh
$
```

Skriv ditt svar her...

Format

Words: 0

Maks poeng: 60

3(a) C og Assembly

Hva slags kode er listet opp i venstre kolonne?

Finn de som passer sammen

	PowerShell	Assembly	Java bytecode	bash	C
ls -l script.sh	○	○	○	○	○
getstatic #18	○	○	○	○	○
Get-ChildItem	○	○	○	○	○
#include <stdio.h>	○	○	○	○	○
add %rdx, %rbx	○	○	○	○	○

Maks poeng: 10

3(b) C og Assembly

Du kompilerer et C-program med **gcc prog.c** Hva inneholder filen **a.out** etterpå?

Velg ett alternativ

- Shell code
- Bytecode
- Maskinkode
- C-kode
- Assembly kode

Maks poeng: 10

3(c) C og Assembly

Du kompilerer et C-program med **gcc -S prog.c** Hva inneholder filen **prog.s** etterpå?

Velg ett alternativ

- Bytecode
- Maskinkode
- Assembly kode
- Shell code
- C-kode

Maks poeng: 10

3(d) C og Assembly

Anta at du har følgende C-program med navn **svar.c** på en Linux-maskin:

```
#include <stdio.h>
int svar = 41;
void enlinje()
{
  svar++;
}
int main ()
{
  printf("Svar = %d\n", svar);
  printf("Kaller enlinje()...\n");
  enlinje();
  printf("Svar = %d\n", svar);
}
```

Forklar kort hvordan du kompilerer og kjører dette programmet i et bash-shell og hva output fra programmet blir når du kjører det.

Skriv ditt svar her...

Format

Words: 0

Maks poeng: 10

3(e) C og Assembly

Du endrer litt på **svar.c** slik at det ser slik ut

```
#include <stdio.h>
int svar = 41;
extern void enlinje();
int main ()
{
  printf("Svar = %d\n", svar);
  printf("Kaller enlinje()...\n");
  enlinje();
  printf("Svar = %d\n", svar);
}
```

og lager i tillegg et program **en.c** som ser slik ut:

```
void enlinje()
{
  extern int svar;
  svar++;
}
```

Forklar kort hva følgende tre kommandoer gjør:

```
$ gcc -c svar.c -o svar
$ gcc -c en.c -o en
$ gcc svar en -o run
```

og hvordan du nå kan kjøre programmet. Restultatet blir eksakt det samme som når du kjørte programmet i forrige oppgave. Forklar kort hvorfor resultatet blir det samme.

Skriv ditt svar her...

Format

Words: 0

Maks poeng: 10

3(f) C og Assembly

Du kjører så kommandoen

$ gcc -S en.c

og det lages da en fil **en.s** med følgende innhold:

```
    .file   "en.c"
    .text
    .globl  enlinje
    .type   enlinje, @function
enlinje:
.LFB0:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    movl    svar(%rip), %eax
    addl    $1, %eax
    movl    %eax, svar(%rip)
    nop
    popq    %rbp
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size   enlinje, .-enlinje
    .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.9) 5.4.0 20160609"
    .section    .note.GNU-stack,"",@progbits
```

Forklar kort hva innholdet av denne filen er og spesielt hva de tre helt sentrale kodelinjene

```
movl    svar(%rip), %eax
addl    $1, %eax
movl    %eax, svar(%rip)
```

gjør. (Merk: **svar(%rip)** er en minnereferanse til der i RAM hvor den globale variabelen **svar** ligger).

Skriv ditt svar her...

Format

Words: 0

Maks poeng: 10

3(g) C og Assembly

Du lager nå en fil **minimal.s** med følgende innhold

```
   .globl   enlinje
enlinje:
   incl   svar(%rip)
   ret
```

og lar den erstatte **en.s** ved å kjøre kommandoen

$ gcc svar minimal.s -o run

Når du nå kjører programmet får du nøyaktig samme resultat. Forklar kort hvorfor **minimal.s** kan erstatte filen **en.s**.

Det du har lært i denne oppgaven er det nyttig å ta med seg til neste oppgave om synkronisering.

Skriv ditt svar her...

Format

Words: 0

Maks poeng: 10

4(a) Synkronisering

*Før du gjør denne oppgaven bør du gjøre forrige oppgave **C og Assembly** da funksjonen **enlinje()** også brukes i denne oppgaven og erfaringene fra den forrige oppgaven er viktige for å løse denne om synkronisering.*

Studer programmet **thread.c** og forklar i korte trekk det vesentligste som skjer når det kjøres.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
int svar = 0;
extern void enlinje();
void *inc()
{
  printf("Starter; svar verdi: %d\n", svar);

  for (int i = 0; i < 10000000; i++)
   {
     enlinje();
   }

  printf("Avslutter; svar verdi: %d\n", svar);
}

int main()
{
        pthread_t thread1, thread2;

        pthread_create( &thread1, NULL, inc,NULL);
        pthread_create( &thread2, NULL, inc, NULL);

        pthread_join( thread1, NULL);
        pthread_join( thread2, NULL);

        printf("Main avslutter; svar verdi: %d\n", svar);
        exit(0);
}
```

Funksjonen **enlinje()** er den samme som i oppgaven om **C og Assembly** og ligger i filen **en.c**:

```
void enlinje()
{
  extern int svar;
  svar++;
}
```

Skriv ditt svar her...

Format | B I U x_2 x^2 I_x | Ω | ∑ | ABC

Words: 0

Maks poeng: 10

4(b) Synkronisering

På en Linux maskin med 8 CPU'er kompilerer du programmene i et bash shell på følgende måte:

```
$ gcc -c en.c -o en
$ gcc -c -pthread thread.c -o thread
$ gcc -pthread en thread -o run
```

Deretter kjører du programmet to ganger og resultatet blir følgende:

```
$ ./run
Starter; svar verdi: 0
Starter; svar verdi: 356150
Avslutter; svar verdi: 10112917
Avslutter; svar verdi: 10300354
Main avslutter; svar verdi: 10300354
$ ./run
Starter; svar verdi: 0
Starter; svar verdi: 618427
Avslutter; svar verdi: 6893160
Avslutter; svar verdi: 10868668
Main avslutter; svar verdi: 10868668
$
```

Du kjører programmet 10 ganger til og får aldri det samme svaret. Forklar kort disse resultatene og hva som kan være årsaken til at resultatene blir forskjellige når samme program kjøres flere ganger.

Skriv ditt svar her...

Format

Words: 0

Maks poeng: 10

4(c) Synkronisering

Du kjører nå programmet på en litt annen måte:

$ **taskset -c 0 ./run**
Starter; svar verdi: 0
Starter; svar verdi: 2441853
Avslutter; svar verdi: 15177160
Avslutter; svar verdi: 15922585
Main avslutter; svar verdi: 15922585
$ **taskset -c 0 ./run**
Starter; svar verdi: 0
Starter; svar verdi: 2599202
Avslutter; svar verdi: 17348681
Avslutter; svar verdi: 20000000
Main avslutter; svar verdi: 20000000
$ **taskset -c 0 ./run**
Starter; svar verdi: 0
Starter; svar verdi: 2592637
Avslutter; svar verdi: 18114661
Avslutter; svar verdi: 16272351
Main avslutter; svar verdi: 16272351
$ **taskset -c 0 ./run**
Starter; svar verdi: 0
Starter; svar verdi: 2975707
Avslutter; svar verdi: 17711781
Avslutter; svar verdi: 20000000
Main avslutter; svar verdi: 20000000
$ **taskset -c 0 ./run**
Starter; svar verdi: 0
Starter; svar verdi: 2803083
Avslutter; svar verdi: 17302934
Avslutter; svar verdi: 20000000
Main avslutter; svar verdi: 20000000

Tre av gangene blir svaret nå 20000000. Forklar hvordan programmet kjøres med denne bruken av **taskset** og hva som er forskjellig fra når man bare kjører det med **./run**. Prøv å forklare hva som kan være årsaken til at svaret nå relativt ofte blir 20000000 men ikke alltid.

Skriv ditt svar her...

Format

Words: 0

Maks poeng: 10

4(d) Synkronisering

Du bruker nå erfaringene fra oppgaven **C og Assembly** og bruker filen **minimal.s**

```
   .globl   enlinje
enlinje:
   incl   svar(%rip)
   ret
```

til å lage en ny versjon av programmet med

$ gcc -pthread thread minimal.s -o run

Når du nå kjører dette programmet, får du hver eneste gang nøyaktig det samme svaret:

$ **taskset -c 0 ./run**
Starter; svar verdi: 0
Starter; svar verdi: 3155197
Avslutter; svar verdi: 17310147
Avslutter; svar verdi: 20000000
Main avslutter; svar verdi: 20000000
$ **taskset -c 0 ./run**
Starter; svar verdi: 0
Starter; svar verdi: 2828311
Avslutter; svar verdi: 18090712
Avslutter; svar verdi: 20000000
Main avslutter; svar verdi: 20000000
$ **taskset -c 0 ./run**
Starter; svar verdi: 0
Starter; svar verdi: 3078129
Avslutter; svar verdi: 17417198
Avslutter; svar verdi: 20000000
Main avslutter; svar verdi: 20000000
$ **taskset -c 0 ./run**
Starter; svar verdi: 0
Starter; svar verdi: 2946892

Avslutter; svar verdi: 18163634
Avslutter; svar verdi: 20000000
Main avslutter; svar verdi: 20000000

Foklar på hvilken måte det resulterende programmet **run** endres når du burker **minimal.s** og hvorfor du nå får det samme resulatet 20000000 hver gang.

Skriv ditt svar her...

Format

Words: 0

Maks poeng: 10

4(e) Synkronisering

Du kjører så programmet laget med **minimal.s** uten taskset på følgende måte:

$ **./run**
Starter; svar verdi: 0
Starter; svar verdi: 613327
Avslutter; svar verdi: 7947390
Avslutter; svar verdi: 11029945
Main avslutter; svar verdi: 11029945
$ **./run**
Starter; svar verdi: 0
Starter; svar verdi: 878021
Avslutter; svar verdi: 9865620
Avslutter; svar verdi: 11007289
Main avslutter; svar verdi: 11007289
$ **./run**
Starter; svar verdi: 0
Starter; svar verdi: 888015
Avslutter; svar verdi: 8496452
Avslutter; svar verdi: 10370608
Main avslutter; svar verdi: 10370608

og du får aldri det samme svaret. Forklar hvordan dette kan være mulig til tross for at du bruker **minimal.s**.

Skriv ditt svar her...

Format

Words: 0

Maks poeng: 10

4(f) Synkronisering

Du erstatter nå **minimal.s** med **lockMinimal.s** som ser slik ut

```
   .globl   enlinje
enlinje:
   lock
   incl   svar(%rip)
   ret
```

og lager **run** på følgende måte:

$ **gcc -pthread thread lockMinimal.s -o run**

Deretter kjører du programmet en rekke ganger

$ **./run**
Starter; svar verdi: 0
Starter; svar verdi: 199526
Avslutter; svar verdi: 19732245
Avslutter; svar verdi: 20000000
Main avslutter; svar verdi: 20000000
$ **./run**
Starter; svar verdi: 0
Starter; svar verdi: 250043
Avslutter; svar verdi: 19744429
Avslutter; svar verdi: 20000000
Main avslutter; svar verdi: 20000000
$ **./run**
Starter; svar verdi: 0
Starter; svar verdi: 219440
Avslutter; svar verdi: 19799083
Avslutter; svar verdi: 20000000
Main avslutter; svar verdi: 20000000

og hver eneste gang du kjører blir svaret nå 20000000. Forklar hva som nå skjer og hvordan endringen fra **minimal.s** til **lockMinimal.s** førte til dette.

Skriv ditt svar her...

Format | B I U x_2 x^2 | I_x | | | | Ω | | | ∑ | ABC |

Words: 0

Maks poeng: 10

4(g) Synkronisering

Studer følgende pseudo-kode som definerer to metoder for å serialisere tråder som kjører samtidig og uavhengig av hverandre:

```
static boolean lock = false; // felles variabel

GetMutex(lock)
{
   while(lock){} // venter til lock blir false
   lock = true;
}

ReleaseMutex(lock)
{
   lock = false;
}
```

Tanken er at metodene skal brukes på følgende måte av tråder som skal kjøre et kritisk avsnitt:

```
GetMutex(lock);
KritiskAvsnitt();
ReleaseMutex(lock);
```

Forklar kort hvordan metodene sørger for serialisering og påvis eventuelt tifeller hvor løsningen ikke fungerer.

Skriv ditt svar her...

Words: 0

Maks poeng: 10

5(a) PowerShell

Hvordan finner du ut hvilke properties og metoder som objektene som returneres av kommandoen **ls** (som egentlig er et alias for Get-Childitem) har?

Velg ett alternativ

- ls | grep Property
- Get-Properties | ls
- ls.GetProperty
- (ls).GetProperty
- ls | Get-Member

Maks poeng: 10

5(b) PowerShell

Studer resultatet av følgende kommandoer utført i et PowerShell:

PS C:\Users\haugerud> **$now = Get-Date**
PS C:\Users\haugerud> **$now - (ls)[0].CreationTime**

```
Days              : 1587
Hours             : 18
Minutes           : 49
Seconds           : 27
Milliseconds      : 73
Ticks             : 1371845670739206
TotalDays         : 1587,78434113334
TotalHours        : 38106,8241872002
TotalMinutes      : 2286409,45123201
```

TotalSeconds : 137184567,073921
TotalMilliseconds : 137184567073,921

Skriv så ned en PowerShell one-liner som gir deg hvor mange dager gammel filen **fil.txt** er. Du kan anta at filen ligger i mappen du står i og resultatet av kommandoen skal være i antall dager med desimaler.

Skriv ditt svar her...

Format

Words: 0

Maks poeng: 10

5(c) PowerShell

I denne oppgaven skal du skrive et PowerShell script med navn **alder.ps1** som når det kjøres gir deg gjennomsnittlig alder for alle filer og mapper i den mappen som scriptet kjøres i. Alderen for en fil er antall dager siden den ble laget, hvor dager er gitt med desimaler.

Når du kjører scriptet skal output tilsvare det som er gitt i følgende eksempel:

```
PS C:\Users\haugerud> .\alder.ps1
Snitt-alder 1364,83201319544 dager for 28 filer og mapper
PS C:\Users\haugerud>
```

Hvis du skulle løse denne oppgaven og hadde tilgang til en web-browser, ville du kanskje google etter "PowerShell foreach" og denne siden er et av de første treffene for dette søket.

Skriv ditt svar her...

Format

Words: 0

Maks poeng: 30

```
$ man echo
ECHO(1)                                                                  User Comma
nds                                                                        ECHO(1)

NAME
       echo - display a line of text

SYNOPSIS
       echo [SHORT-OPTION]... [STRING]...
       echo LONG-OPTION

DESCRIPTION
       Echo the STRING(s) to standard output.

       -n     do not output the trailing newline

       -e     enable interpretation of backslash escapes

       -E     disable interpretation of backslash escapes (default)

       --help display this help and exit

       --version
              output version information and exit

       If -e is in effect, the following sequences are recognized:

       \\     backslash

       \a     alert (BEL)

       \b     backspace

       \c     produce no further output

       \e     escape

       \f     form feed

       \n     new line

       \r     carriage return

       \t     horizontal tab

       \v     vertical tab

       \0NNN  byte with octal value NNN (1 to 3 digits)

       \xHH   byte with hexadecimal value HH (1 to 2 digits)

       NOTE:  your  shell may have its own version of echo, which usually supersedes the ve
rsion described here.  Please refer to your shell's documentation for details about
       the options it supports.

AUTHOR
       Written by Brian Fox and Chet Ramey.

REPORTING BUGS
       GNU coreutils online help: <http://www.gnu.org/software/coreutils/>
       Report echo translation bugs to <http://translationproject.org/team/>

COPYRIGHT
       Copyright (C) 2016 Free Software Foundation, Inc.  License GPLv3+: GNU GPL version 3
 or later <http://gnu.org/licenses/gpl.html>.
       This is free software: you are free to change and redistribute it.  There is NO WARR
ANTY, to the extent permitted by law.
```

```
SEE ALSO
       Full documentation at: <http://www.gnu.org/software/coreutils/echo>
       or available locally via: info '(coreutils) echo invocation'

GNU coreutils 8.25                                                    February 2
016                                                                   ECHO(1)
```

```
$ help for
for: for NAME [in WORDS ... ] ; do COMMANDS; done
    Execute commands for each member in a list.

    The `for' loop executes a sequence of commands for each member in a
    list of items.  If `in WORDS ...;' is not present, then `in "$@"' is
    assumed.  For each element in WORDS, NAME is set to that element, and
    the COMMANDS are executed.

    Exit Status:
    Returns the status of the last command executed.
```

```
$help while
while: while COMMANDS; do COMMANDS; done
    Execute commands as long as a test succeeds.

    Expand and execute COMMANDS as long as the final command in the
    `while' COMMANDS has an exit status of zero.

    Exit Status:
    Returns the status of the last command executed.
```

```
$help read
read: read [-ers] [-a array] [-d delim] [-i text] [-n nchars] [-N nchars] [-p prompt] [-t t
imeout] [-u fd] [name ...]
    Read a line from the standard input and split it into fields.

    Reads a single line from the standard input, or from file descriptor FD
    if the -u option is supplied.  The line is split into fields as with word
    splitting, and the first word is assigned to the first NAME, the second
    word to the second NAME, and so on, with any leftover words assigned to
    the last NAME.  Only the characters found in $IFS are recognized as word
    delimiters.

    If no NAMEs are supplied, the line read is stored in the REPLY variable.

    Options:
      -a array  assign the words read to sequential indices of the array
                variable ARRAY, starting at zero
      -d delim  continue until the first character of DELIM is read, rather
                than newline
      -e                use Readline to obtain the line in an interactive shell
      -i text   Use TEXT as the initial text for Readline
      -n nchars return after reading NCHARS characters rather than waiting
                for a newline, but honor a delimiter if fewer than NCHARS
                characters are read before the delimiter
      -N nchars return only after reading exactly NCHARS characters, unless
                EOF is encountered or read times out, ignoring any delimiter
      -p prompt output the string PROMPT without a trailing newline before
```

```
            attempting to read
      -r                do not allow backslashes to escape any characters
      -s                do not echo input coming from a terminal
      -t timeout        time out and return failure if a complete line of input is
            not read within TIMEOUT seconds.  The value of the TMOUT
            variable is the default timeout.  TIMEOUT may be a
            fractional number.  If TIMEOUT is 0, read returns immediately,
            without trying to read any data, returning success only if
            input is available on the specified file descriptor.  The
            exit status is greater than 128 if the timeout is exceeded
      -u fd             read from file descriptor FD instead of the standard input

   Exit Status:
   The return code is zero, unless end-of-file is encountered, read times out
   (in which case it's greater than 128), a variable assignment error occurs,
   or an invalid file descriptor is supplied as the argument to -u.
```

ForEach (loop statement)

Loop through a set of input objects and perform an operation (execute a block of statements) against each.

Syntax

```
      ForEach (item In collection) {ScriptBlock}
```

key

```
   item         A variable to hold the current item
   collection   A collection of objects e.g. filenames, registry keys, servernames
   ScriptBlock  A block of script to run against each object.
```

The collection will be evaluated and stored as an array in memory before the scriptblock is executed.

The foreach statement does not use pipelining (unlike ForEach-Object) If you use foreach in a command pipeline PowerShell will actually run the foreach alias that calls ForEach-Object.

Use the ForEach statement when the collection of objects is small enough that it can be loaded into memory.
Use the ForEach-Object cmdlet when you want to pass only one object at a time through the pipeline, minimising memory usage. In most cases ForEach will run faster than ForEach-Object, there are exceptions, such as starting multiple background jobs. If in doubt test both options with Measure-Command.

In PowerShell 4.0 and later, the ForEach method provides even faster performance.

Examples

Loop through an array of strings:

```
 $trees = @("Alder","Ash","Birch","Cedar","Chestnut","Elm")

 foreach ($tree in $trees) {
   "$tree = " + $tree.length
 }
```

Loop through a collection of the numbers, echo each number unless the number is 2:

```
 foreach ($num in 1,2,3,4,5) {
  if ($num -eq 2) { continue } ; $num
 }
```

Loop through a collection of .txt files:

```
  foreach ($file in get-ChildItem *.txt) {
    Echo $file.name
  }
```

"The only way to do great work is to love what you do. If you haven’t found it yet, keep looking. Don’t settle. As with all matters of the heart, you’ll know when you find it" - Steve Jobs

Related:

$foreach variable - Refers to the enumerator in a foreach loop.
Break statement
Continue statement
ForEach-Object - Loop for each object in the pipeline (foreach)
ForEach (method) - Loop through items in a collection
For - Loop through items that match a condition
IF - Conditionally perform a command
Switch - Multiple if statements
While - Loop while a condition is True