

Εργαστήριο Λειτουργικών Συστημάτων

Μάθημα 6^{ου} Εξαμήνου.

Τομέας Λογικού και Υπολογιστών

Ιωάννης Χατζηγιαννάκης

Σημειώσεις Μαθήματος
Ενότητα 2



Σημειώσεις Μαθήματος – 2^η Ενότητα

Περιβάλλον Προγραμματισμού UNIX

Κέλυφος Bash

Μεταβλητές Περιβάλλοντος

Shell Scripts

Επεξεργαστής Ροών Κειμένου – sed

Γλώσσα Διαχείρισης Κειμένων – awk



Γενικά

- ▶ Το κέλυφος (shell) προσφέρει ένα εναλλακτικό περιβάλλον από την κονσόλα
 - ▶ Επιτρέπει τον συνδυασμό εντολών με την χρήση script
 - ▶ Προσφέρει εναλλακτικούς τρόπους για την επίτευξη σύνθετων ενεργειών
 - ▶ Επιτρέπει αποθήκευση μεταβλητών
- ▶ Υπάρχουν πολλά διαφορετικά κέλυφη *kom*, *tcsh*, *zsh* ...
- ▶ Κάθε χειριστής έχει ένα προεπιλεγμένο κέλυφος
 - ▶ Η επιλογή διατηρείται στο αρχείο */etc/passwd*
`ichatz:x:1000:1000:::/home/ichatz:/bin/bash`
 - ▶ Η εντολή *chsh* αλλάζει το κέλυφος
- ▶ Προσφέρει αρχεία ρυθμίσεων διαφορετικά για κάθε λογαριασμό



Παράδειγμα Script στο κέλυφος BASH

```
$ for dir in $PATH
>do
> if [ -x $dir/gcc ]
> then
>     echo Found $dir/gcc
>     break
> else
>     echo Searching $dir/gcc
> fi
>done
```

- ▶ Για κάθε φάκελο που ορίζεται στην μεταβλητή περιβάλλοντος \$PATH
- ▶ Έλεγξε αν περιέχει το εκτελέσιμο αρχείο gcc
 - ▶ Αν υπάρχει εκτύπωσε το *path* και σταμάτα
 - ▶ Αλλιώς συνέχισε την αναζήτηση στον επόμενο φάκελο



Παράδειγμα Script στο κέλυφος CSH/TCSSH

```
$ foreach dir ( $path )
>   if ( -x $dir/gcc ) then
>     echo Found $dir/gcc
>     break
>   else
>     echo Searching $dir/gcc
>   endif
> end
```

- ▶ Μοιάζει περισσότερο με C
- ▶ Θα εισάσουμε στο κέλυφος BASH
 - ▶ Είναι βασισμένο πάνω στο κέλυφος Bourne
 - ▶ Είναι ανοικτού κώδικα λογισμικό
 - ▶ Για να το χρησιμοποιήσετε εκτελέστε την εντολή: `bash`



Γραμμή Εντολών

```
# bash
bash-3.00#
```

- ▶ Μοιάζει με την γραμμή εντολών της κονσόλας
 - ▶ Το αριστερό μέρος μπορεί να αλλάξει
 - ▶ Το δεξί μέρος χρησιμοποιείτε για εκτέλεση εντολών
- ▶ Προσφέρει κάποιες ενσωματωμένες εντολές
 - ▶ Υλοποιημένες μέσα στον κώδικα του κελύφους
 - ▶ Οι εντολές αυτές εκτελούνται από την διεργασία του κελύφους
- ▶ Προσφέρει δυνατότητες εκτέλεσης script
 - ▶ Για αυτό τον λόγο το ονομάζουμε **περιβάλλον προγραμματισμού**



Ενσωματωμένες Εντολές

Εντολή	Περιγραφή	Παράδειγμα
cd	Αλλαγή φακέλου	cd ..
declare	Ορίζει μια μεταβλητή	declare myvar
echo	Εμφάνιση κειμένου στην βασική έξοδο	echo hello
exec	Αντικατάσταση του bash από μια άλλη διεργασία	exec ls
exit	Τερματισμός του κελύφους	exit
export	Ορισμός καθολικής μεταβλητής	export myvar=1
history	Εμφάνιση ιστορικού εντολών	history
kill	Αποστολή σήματος σε μια διεργασία	kill 1121
let	Υπολογισμός μιας αριθμητικής πράξης	let myvar=3+5



Ενσωματωμένες Εντολές

Εντολή	Περιγραφή	Παράδειγμα
local	Ορισμός τοπικής μεταβλητής	local myvar=5
pwd	Εμφάνιση τρέχοντος φακέλου	pwd
read	Ανάγνωση από την βασική είσοδο σε μια μεταβλητή	read myvar
readonly	Κλειδώνει μια μεταβλητή	readonly myvar
return	Ολοκλήρωση μιας συνάρτησης και επιστροφή τιμής	return 1
set	Εμφάνιση μεταβλητών	set
shift	Μεταθέτει τις παραμέτρους	shift 2
test	Έλεγχος μιας έκφρασης	test -d temp
trap	Παρακολούθηση ενός σήματος	trap "echo Signal" 3



Πρώθηση Εισόδου / Εξόδου

- ▶ Οι εντολές παράγουν έξοδο – χρησιμοποιούμε το επίθεμα `>` για την πρώθηση σε κάποιο αρχείο
 - # `ls > filelist`
 - ▶ Θα δημιουργηθεί ένα νέο αρχείο με όνομα `filelist`
 - ▶ Αν υπάρχει ήδη, το νέο αρχείο θα αντικαταστήσει το παλιό
 - ▶ χρησιμοποιούμε το επίθεμα `>>` για την πρώθηση σε κάποιο υπάρχον αρχείο
 - # `ls -lt /root/doc >> /root/filelist`
- ▶ Οι εντολές απαιτούν είσοδο – χρησιμοποιούμε το επίθεμα `<` για την πρώθηση ενός αρχείου ως είσοδο
 - # `sort < /root/filelist`
- ▶ Για να προωθήσουμε την έξοδο μιας εντολής στην είσοδο μιας άλλης – χρησιμοποιούμε το επίθεμα `|`
 - # `who | sort --ταξινόμηση καταλόγου χειριστών`
 - # `ls /root | grep rc | wc -l --καταμέτρηση αρχείων 'rc'`



Διεργασίες

- ▶ Μπορούμε να εκτελέσουμε εντολές **σειριακά** διαχωρίζοντας τις εντολές με `;`
 - ▶ Εκτελούνται όλες οι εντολές και **όταν ολοκληρωθεί και η τελευταία**, προσφέρεται νέο prompt
 - # `who | sort ; date`
- ▶ Μπορούμε να εκτελέσουμε εντολές **παράλληλα** διαχωρίζοντας τις εντολές με `&`
 - ▶ Εκτελούνται όλες οι εντολές και **προσφέρεται άμεσα** νέο prompt
 - # `pr junk | lpr &`
- ▶ Η εκτέλεση μια εντολής είναι μια **διεργασία**
 - ▶ Η εντολή `ps` εμφανίζει τις τρέχουσες διεργασίες
 - ▶ Η εντολή `wait` περιμένει μέχρι να ολοκληρωθούν όλες οι εντολές που εκτελέστηκαν με `&`



Κατάλογος Διεργασιών

```
# ps -a
  PID TTY   TIME CMD
  106  c1   0:01 -sh
  4114 co   0:00 /bin/sh /usr/bin/packman
  2114 co   0:00 -sh
  6762 c1   0:00 ps -a
     87  c2   0:00 getty
     90  c3   0:00 getty
```

- ▶ Παράμετρος `a` – εμφάνιση διεργασιών που δημιουργήθηκαν από κονσόλες
- ▶ Σήλη `PID` – μοναδική ταυτότητα διεργασίας
- ▶ Σήλη `TTY` – κονσόλα που δημιούργησε την διεργασία
- ▶ Σήλη `TIME` – συνολικός χρόνος εκτέλεσης
- ▶ Σήλη `CMD` – εντολή που εκτελέστηκε



Εργαλεία διαχείρισης διεργασιών

- ▶ Τερματισμός διεργασίας – εντολή `kill (PID)`
- ▶ Μπορούμε να εκτελέσουμε μια εντολή με διαφορετική προτεραιότητα
 - ▶ πρόβλημα `nice`
 - # `nice pr junk | lpr &`
- ▶ Μπορούμε να καθυστερήσουμε την εκτέλεση μιας εντολής
 - ▶ πρόβλημα `at`

```
# at 1500
ls -l / /root /dir | wc > allfiles
pr allfiles | lpr ; date > lpr-endtime &
date > lpr-starttime
^D
at: /usr/spool/at/07.111.1500.67 created
#
```



Εντολή echo (1)

- ▶ Βασικός τρόπος για την δημιουργία εξόδου
- ▶ Εκτυπώνει τις τιμές των μεταβλητών
- ▶ Αναγνωρίζει κάποιους ειδικούς χαρακτήρες (ή μετα-χαρακτήρες)

```
bash-3.00# echo hello there
hello there
bash-3.00# let myvar=1; echo $myvar
1
bash-3.00# echo *
junk lpr-starttime temp
bash-3.00# echo print '*' "don't"
print * don't
```



Εντολή echo (2)

- ▶ Μπορεί να περιέχει περισσότερες από μία γραμμές
- ▶ Μπορεί να εκτελέσει εντολές

```
bash-3.00# echo 'hello
there'
hello
there
bash-3.00# echo hello\
there
hello there
bash-3.00# echo `date`
Mon Apr 30 16:12:21 GMT 2007
bash-3.00# echo -n `date` " "
Mon Apr 30 16:12:21 GMT 2007 bash-3.00#
```



Ειδικοί Χαρακτήρες – Μετα-χαρακτήρες

- ▶ Χαρακτήρας ? – ένας χαρακτήρας, π.χ.
ls /etc/rc.????
- ▶ Χαρακτήρας * – πολλοί χαρακτήρες, π.χ.
ls /etc/rc.*
- ▶ Πίνακας (...) – συγκεκριμένοι χαρακτήρες, π.χ.
ls [abc].c
- ▶ Η χρήση των χαρακτήρων αναζήτησης ονομάζεται και filename substitution
- ▶ Μπορούμε να τους χρησιμοποιήσουμε σε συνδυασμό με όλες τις εντολές
- ▶ Προσοχή η παρακάτω εντολή δεν λειτουργεί
mv *.x *.y



Μεταβλητές Περιβάλλοντος

- ▶ Το κέλυφος επιτρέπει τον ορισμό μεταβλητών
- ▶ Οι αρχικές τιμές των μεταβλητών ορίζονται στο αρχείο ρυθμίσεων του συστήματος και του συγκεκριμένου λογαριασμού
- ▶ Οι τιμές των μεταβλητών ισχύουν έως το τέλος του session
 - ▶ Μέχρι να τις διαγράψει ο χειριστής
- ▶ Οι μεταβλητές με ΚΕΦΑΛΑΙΑ γράμματα είναι **καθολικές** – μεταφέρονται σε όλα τα κελύφη που θα ξεκινήσουν από το τρέχον
- ▶ Οι μεταβλητές με μικρά γράμματα είναι **τοπικές** – ισχύουν μόνο για το τρέχον κέλυφος

```
HOME          # The path to your home directory
term          # The terminal type
```



Μεταβλητές Περιβάλλοντος

- ▶ Μπορούμε να χρησιμοποιούμε τις μεταβλητές από την γραμμή εντολών
- ▶ Χρησιμοποιούμε τον τελεστή \$

```
bash-3.00# myvar="hello"; echo $myvar
hello
bash-3.00# myvar="ls -la"
bash-3.00# $myvar
lrwxrwxrwx 1 bin operator 2880 Jun 1 1993 b
-r--r--r-- 1 root operator 448 Jun 1 1993 b
drwxr-sr-x 2 root operator 11264 May 11 17:00 d
...
```



Ειδικές Μεταβλητές

- ▶ Ορισμένες μεταβλητές έχουν ειδική σημασία

Όνομα	Περιγραφή
USER	Όνομα λογαριασμού χειριστή
HOME	Προσωπικός φάκελος χειριστή
TERM	Τύπος τερματικής συσκευής
SHELL	Ονομασία κελύφους
PATH	Λίστα φακέλων με εκτελέσιμες εντολές
MANPATH	Λίστα φακέλων με σελίδες βοήθειας
PWD	Ενεργός φάκελος
OLDPWD	Προηγούμενος ενεργός φάκελος
HOSTNAME	Ονομασία συστήματος



Χειρισμός Μεταβλητών

- ▶ Οι εντολές *env*, *printenv* εμφανίζουν την λίστα με τις ΚΑΘΟΛΙΚΕΣ μεταβλητές
- ▶ Η εντολή *set* εμφανίζει την λίστα με τις ΤΟΠΙΚΕΣ μεταβλητές
- ▶ Για να ορίσουμε μια ΚΑΘΟΛΙΚΗ μεταβλητή χρησιμοποιούμε την εντολή *export*
- ▶ Δήλωση μεταβλητών σύμφωνα με το περιεχόμενο
 - ▶ String variables - `myvar = "value"`
 - ▶ Integer variables - `declare -i myvar`
 - ▶ Constant variables - `readonly me="ichatz"`
 - ▶ Array variables - `declare -a MYARRAY`
`MYARRAY[0]="one"; MYARRAY[1]=5; echo ${MYARRAY[*]}`
- ▶ Τα ονόματα των μεταβλητών είναι case-sensitive
- ▶ Η εντολή *unset* διαγράφει μια μεταβλητή



Τοπικές - Καθολικές Μεταβλητές

- ▶ Για να ορίσουμε μια ΚΑΘΟΛΙΚΗ μεταβλητή χρησιμοποιούμε την εντολή *export*

```
bash-3.00# myvar="hello"
bash-3.00# set | grep myvar
myvar=hello
bash-3.00# bash --- 2nd Shell
bash-3.00# set | grep myvar
bash-3.00# exit --- End of 2nd
bash-3.00# export myvar="hello"
bash-3.00# set | grep myvar
myvar=hello
bash-3.00# bash --- 2nd Shell
bash-3.00# set | grep myvar
myvar=hello
```



Δημιουργία Νέων Εντολών

- ▶ Μπορούμε να δημιουργήσουμε νέες εντολές
 - ▶ Σε ένα αρχείο κειμένου εισάγουμε τις εντολές
- ▶ Εκτελούμε την νέα εντολή με την χρήση του sh
 - ▶ Ορίζουμε πρόσβαση εκτέλεσης στο αρχείο και το 'καλούμε' απευθείας

```
bash-3.00# echo 'who | wc -l' > nu
bash-3.00# cat nu
who | wc -l
bash-3.00# sh nu
1
bash-3.00# chmod a+x nu
bash-3.00# nu
1
```



Χειρισμός Παραμέτρων (1)

- ▶ Μπορούμε να περάσουμε παραμέτρους σε ένα script
 - ▶ Ονομάζονται command-line arguments
- ▶ Χρησιμοποιούμε τις παραμέτρους σαν μεταβλητές

Παράμετρος	Περιγραφή
\$0	Το όνομα του script
\$1 ... \$9	Η τιμή της 1ης ... 9ης παράμετρου
\$#	Πλήθος παραμέτρων
\$	Όλοι οι παράμετροι σαν string

```
bash-3.00# cat nu
echo Files found: `ls -la $1* | wc -l` "$(\*)"
bash-3.00# nu /b
Files found: 57 (/b*)
```



Χειρισμός Παραμέτρων (2)

- ▶ Για να χειριστούμε περισσότερες από 9 παραμέτρους
 - ▶ Δεν μπορούμε να χρησιμοποιήσουμε \$10
- ▶ Χρησιμοποιούμε την εντολή *shift x*
 - ▶ Μεταφέρει τις παραμέτρους προς τα αριστερά κατά x θέσεις
 - ▶ Προσοχή - οι παλιές παράμετροι χάνονται

```
bash-3.00# cat ten
shift 10
echo $1
echo $* " -- " $#
bash-3.00# ten 1 2 3 4 5 6 7 8 9 10
10
10 -- 1
```



Είσοδος από τον χειριστή

- ▶ Μπορούμε να ζητήσουμε είσοδο με την χρήση της εντολής *read*
 - ▶ Η σύνταξη είναι *read var-name*
 - ▶ μπορούμε να ζητήσουμε πολλαπλές μεταβλητές
read var1 var2 ...
 - ▶ μπορούμε να εμφανίσουμε ένα μήνυμα πριν ζητήσουμε είσοδο
read -p "Enter value:" var

```
bash-3.00# read -p "Enter values:" i j k;\
echo i=$i, j=$j, k=$k
abc d e f
i = abc, j = d, k = e f
```



- ▶ Δυνατότητα μαθηματικών εκφράσεων με ακέραιους
 - ▶ Σχεδόν όπως στην C
 - ▶ Δεν χρειάζεται να έχουμε δηλώσει ότι η μεταβλητή είναι integer
 - ▶ Χρησιμοποιούμε την `expr` αντί για `atoi`

```
(( a = a + 1 ))
a=$(( a+1 ))
a=$(( $a+1 ))
let a = a + 1
let a++
a='expr $a + 1'
```



```
if [ condition 1 ]; then
    if [[ condition 2 && condition 3 ]]; then
        ...
    fi
elif [ condition 4 ] || [ condition 5 ]; then
    ...
else
    ...
fi
```

- ▶ Η εντολή `test` επιτρέπει την αποτίμηση μια έκφρασης
 - ▶ Επιστρέφει `true` ή `false`
 - ▶ Προσφέρει μεγάλο εύρος εκφράσεων
 - ▶ π.χ., αν έχουμε δικαιώματα αλλαγής σε ένα αρχείο


```
if test -w "$1"; then echo "File $1 is writable" fi
```



Τελεστής	Περιγραφή
<code>-gt</code>	Μεγαλύτερο ή ίσο
<code>-ge</code>	Μεγαλύτερο
<code>-lt</code>	Μικρότερο
<code>-le</code>	Μικρότερο ή ίσο
<code>-eg</code>	Ισο
<code>-ne</code>	Διάφορο
<code>-n str</code>	Μέγεθος <code>string</code> μεγαλύτερο του 0
<code>-z str</code>	Μηδενικό <code>string</code>
<code>-d file</code>	Το <code>file</code> είναι φάκελος
<code>-s file</code>	Το <code>file</code> δεν έχει μηδενικό μέγεθος
<code>-f file</code>	Το <code>file</code> υπάρχει
<code>-r file</code>	Έχουμε πρόσβαση ανάγνωσης για το <code>file</code>
<code>-w file</code>	Έχουμε πρόσβαση εγγραφής για το <code>file</code>
<code>-x file</code>	Έχουμε πρόσβαση εκτέλεσης για το <code>file</code>



```
bash-3.00# cat check.sh
#!/bin/bash
read -p "Enter a filename: " filename
if [ ! -w "$filename" ]; then
    echo "File is not writable"
    exit 1

elif [ ! -r "$filename" ] ; then
    echo "File is not readable"
    exit 1

fi
...
```



Παράδειγμα Συνθήκης Ελέγχου και test (2)

```
bash-3.00# cat check.sh
#!/bin/bash
TMPFILE = "diff.out"

diff $1 $2 > $TMPFILE

if [ ! -s "$TMPFILE" ]; then
    echo "File are the same"
else
    more $TMPFILE
fi

if [ -f "$TMPFILE" ]; then
    rm -rf $TMPFILE
fi
```

Τελεστές boolean

```
if [ condition 1 && condition a]; then
    if [ condition 2 || condition b]; then
        ...
    fi
elif [ ! condition 3 ] ; then
    ...
else
    ...
fi
```

Συνθήκη Ελέγχου – case

```
case STRING in
    pattern 1 )
        ... ;;
    pattern 2 | pattern 3)
        ... ;;
*)
    echo "None of the above";
    ...
esac
```

Παράδειγμα Ελέγχου – case

```
#!/bin/bash
read -p "Enter command: " command
case $command in
    all | ALL )
        echo "Display all files..."
        ls -la;;
    list | LIST)
        echo "Display all non-hidden files ..."
        ls -l;;
*)
    echo "Invalid choise"
    ls;;
esac
```

Βρόχος – for

```
for VAR in <list>
do
...
done
```

```
for i in 6 3 1 2
do
echo $i
done | sort -n
```

```
for i in *.c
```



Βρόχος – while

```
while [ expression ];
do
...
done
```

```
i = 1
while [[ $i -lt 10 ]];
do
echo $i
((i++))
done
```



Βρόχος – until

```
until [ expression ];
do
...
done
```

```
Stop = "N"
until [[ $Stop = "Y" ]];
do
ps -ef
read -p "Do you want to stop? (Y/N)" Stop
done
echo "Stopping..."
```



Συναρτήσεις

```
function name [()]
{
...
[return]
}
```

- ▶ Όλες οι συναρτήσεις πρέπει να οριστούν στην αρχή του script
- ▶ Μπορεί να μην έχει παραμέτρους
- ▶ Οι παράμετροι και η τιμή που επιστρέφουν μπορεί να είναι από οποιοδήποτε τύπο
- ▶ Οι μεταβλητές που ορίζονται μέσα στην συνάρτηση είναι **καθολικές!**
 - ▶ Πρέπει να δηλώσουμε ότι είναι **local**



Παράδειγμα Συναρτήσεων

```
#!/bin/bash
outside = "a global variable"

function mine() {
    local inside="this is local"
    echo $outside
    echo $inside
    outside = "a global with new value"
}

echo $outside
mine
echo $outside
echo $inside
```



Ενα P2P σύστημα ανταλλαγής αρχείων

```
#!/bin/sh
# uP2P.sh 0.0.1, 436 characters (excluding comments)
[ $3 ]&&export W=$1 H="$2 $3" K="mktemp";Z=/dev/null;e(){ echo "$*";};n(){
nc $* >>$Z};k(){ nc -lp $H$* } -e $1 >$Z <$Za;f(){ cat $K|while read h;do
e SW $1 "$2" |n $h;done ;};case $# in 4) SW a "$4"|n $H|while read h p f; do
e SW g "$f"|n $h $p"$f";done;;5) e $H>$K;e SW d $H |n $4 $5>>$K;x $0;;0) x $0
read w c r;| SW = $w |&case $c in a) f 1 "$r";|g) cat "$r";|a) e r>>$K;;d) cat $K
f a "$r";|);|ls|grep "$r"|sed "s/"$/" /";|&&aac;|&&aac
```

- ▶ Για τον πρώτο server
uP2P.sh oslab machineA.ip 9500 machineA.ip 9500
- ▶ Για τον δεύτερο server
uP2P.sh oslab machineB.ip 9500 machineA.ip 9500
- ▶ Για να γίνεται download
uP2P.sh oslab machineA.ip 9500 "pdf"



Γενικά για τον sed (1)

sed: Stream Editor:

- ▶ Είσοδος από αρχείο ή pipe - έξοδος αντίστοιχα
- ▶ Φιλτράρει και επεξεργάζεται το κείμενο εισόδου και το δίνει στον έξοδο, τροποποιημένο κατά τον τρόπο που επισημάνθηκε και δίκως ισχυρή επανατροφοδότηση: συνήθως με κάποια εύρεση pattern και αντικατάσταση.
- ▶ Επεξεργάζεται αρχεία κειμένου ανά γραμμές
- ▶ Τον χρησιμοποιούμε όταν η γνώση για την πληροφορία δεν βρίσκεται τόσο στη δομή του αρχείου (γραμμές, στήλες) όσο στο κείμενο του



Γενικά για τον sed (2)

Το κέλυφος έρχεται με ισχυρά εργαλεία για τη διαχείριση κειμένου

- ▶ Η εντολή sed, την προτιμάμε για απλές πράξεις, όπως
 - ▶ αντικατάσταση ή διαγραφή patterns
 - ▶ όταν η πληροφορία της εισόδου δεν είναι τόσο στη μορφοποίηση της όσο στο κείμενο (γραμμές,) patterns.
- ▶ Είσοδος από αρχείο ή pipe - έξοδος αντίστοιχα
- ▶ Φιλτράρει και επεξεργάζεται το κείμενο εισόδου και το δίνει στον έξοδο, τροποποιημένο κατά τον τρόπο που επισημάνθηκε και δίκως ισχυρή επανατροφοδότηση: συνήθως με κάποια εύρεση pattern και αντικατάσταση.
- ▶ Επεξεργάζεται αρχεία κειμένου ανά γραμμές
- ▶ Τον χρησιμοποιούμε όταν η γνώση για την πληροφορία δεν βρίσκεται τόσο στη δομή του αρχείου (γραμμές, στήλες) όσο στο κείμενο του



pattern space = το υφιστάμενο κείμενο όπως επεξεργάζεται (data buffer)

```
while (readline) {
```

1. διαβάζει μια γραμμή εισόδου από το προκαθορισμένο ρεύμα εισόδου ή από αρχείο, μια-μια κάθε φορά, στο χώρο προτύπων
2. σε κάθε γραμμή, ο sed εκτελεί μια σειρά από εντολές επεξεργασίας (που έχουν γραφτεί από το χρήστη) επί του pattern space
3. γράφει το χώρο προτύπων στο ρεύμα εξόδου

```
}
```



```
sed <options> ' <address><command>'
```

1. address: ένας αριθμός γραμμής του κειμένου εισόδου ή ένα πρότυπο που εμπεριέχεται σε slashes (/pattern/). Καθορίζει που θα εφαρμοστεί η εντολή, σε ποιές γραμμές - αν δεν ορίζεται, η sed θα εφαρμοστεί σε κάθε γραμμή της εισόδου
2. Το πρότυπο περιγράφεται με regular expressions, όπως η grep
3. Εάν δοθούν δυο διευθύνσεις χωρισμένες με κόμμα (,), η εντολή λειτουργεί στο πεδίο γραμμών μεταξύ της πρώτης και δεύτερης διεύθυνσης, συμπεριλαμβανομένων των ιδίων
4. ! = NOT (εφαρμογή στις διευθύνσεις εκτός αυτών που περιγράφηκαν)



a\	Προσθήκη κειμένου μετά την τρέχουσα γραμμή
c\	Αλλαγή του τρέχοντος κειμένου σε «νέο κείμενο»
d	Διαγραφή κειμένου
i\	Προσθήκη κειμένου πριν την τρέχουσα γραμμή
p	Εκτύπωση κειμένου
r	Ανάγνωση αρχείου
s	Αναζήτηση και αντικατάσταση κειμένου
w	Εγγραφή σε αρχείο
-e	Για τον προσδιορισμό πολλαπλών εντολών
-f SCRIPT_FILE	Ορισμός αρχείου sed εντολών
-n	Εκτυπώνονται μόνο οι p εντολές



Η πλέον συνήθης εντολή:

```
sed s 'pattern/replacement/<flags>'
```

- ▶ **pattern**: πρότυπο που αναζητούμε
- ▶ **replacement**: το string που θα αντικαταστήσει το πρότυπο
- ▶ **flags** (προαιρετικά):
 - ▶ **n** (number): ποια τον αριθμό εμφάνιση του προτύπου θα αντικατασταθεί
 - ▶ **g** (global): αντικατάσταση όλων (όχι μόνο της πρώτης εμφάνισης) των προτύπων που εμφανίζονται στην τρέχουσα γραμμή
 - ▶ **p** (print): εκτύπωση του περιεχομένου του pattern space



Αρχείο προς παραδειγματισμό!

```
bash-3.1$ cat -n example.sed
```

```
1 This is the first line of an example text.
2 It is a text with errors.
3 Lots of errors.
4 So much errors, all these errors are making me sick.
5 This is a line not containing any errors.
6 This is the last line.
```



Παραδείγμα 1

```
bash-3.1$ sed 's/erors/errors/g' example.sed
```

```
This is the first line of an example text.
It is a text with errors.
Lots of errors.
So much errors, all these errors are making me sick.
This is a line not containing any errors.
This is the last line.
```

Τι θα συμβεί αν στην εντολή αντικαταστήσουμε το g με τον αριθμό 2; Τι αν το απομακρύνουμε εντελώς;



Παραδείγμα 2

^ Αρχή γραμμής - \$ Τέλος γραμμής

```
bash-3.1$ sed 's/^/> //' example.sed
```

```
> This is the first line of an example text.
> It is a text with errors.
> Lots of errors.
> So much errors, all these errors are making me sick.
> This is a line not containing any errors.
> This is the last line.
```

Τι θα συμβεί αν στην εντολή αντικαταστήσουμε το ^ με \$;



Παραδείγμα 3

```
bash-3.1$ sed -e 's/erors/errors/g' -e
's/last/final/g' example.sed
(or, alternatively)
sed 's/erors/errors/g; s/last/final/g' example.sed
```

```
This is the first line of an example text.
It is a text with errors.
Lots of errors.
So much errors, all these errors are making me sick.
This is a line not containing any errors.
This is the final line.
```



Άλλοι ειδικοί χαρακτήρες

- ▶ `_` ή (κόμμα) Μπορούν να πάρουν τη θέση του / για αναγνωσιμότητα
- ▶ `\:` escape character
- ▶ `&` Αντιστοιχεί στο πρότυπο που έχει βρεθεί (αναφερόμαστε πάντα στην τρέχουσα γραμμή)
- ▶ Προσοχή στο τι αποτελεί σύμβολο regular expression και τι όχι



Παραδείγμα 4

```
bash-3.1$ sed 's/[^ ][^ ]*/(&)/' example.sed
```

```
(This) is the first line of an example text.  
(It) is a text with errors.  
(Lots) of errors.  
(So) much errors, all these errors are making me sick.  
(This) is a line not containing any errors.  
(This) is the last line.
```

Τι θα συνέβαινε αν το pattern ήταν

```
[a-z]\+\. ?  
s/[^ ] ?
```



Παραδείγμα 5

Εκτύπωση μόνο των γραμμών που υπακούουν στο πρότυπο μετά την αλλαγή τους κατά τους όρους της αντικατάστασης που ορίστηκε:

```
bash-3.1$ sed -n 's/errors/errors/gp' example.sed
```

```
It is a text with errors.  
Lots of errors.  
So much errors, all these errors are making me sick.
```

Τι θα συνέβαινε αν εδν υπήρχε ένα ! πριν το p (print)?



Εστίαση σε συγκεκριμένες γραμμές

Μπορούμε να εστιάσουμε σε συγκεκριμένες γραμμές τις αλλαγές μας, δηλώνοντας τις γραμμές με τον αριθμό τους.

```
bash-3.1$ sed '1,3 s/errors/errors/g' example.sed
```

```
This is the first line of an example text.  
It is a text with errors.  
Lots of errors.  
So much errors, all these errors are making me sick.  
This is a line not containing any errors.  
This is the last line.
```



Εστίαση σε συγκεκριμένες γραμμές

Το ίδιο μπορούμε να κάνουμε δίνοντας ένα κοινό τους pattern

```
bash-3.1$ sed '/^T/ s/\ is/\ was/g' example.sed
```

```
This was the first line of an example text.  
It is a text with errors.  
Lots of errors.  
So much errors, all these errors are making me sick.  
This was a line not containing any errors.  
This was the last line.
```

Τι θα συνέβαινε αν εφαρμόζαμε την ακόλουθη εντολή σε ένα αρχείο C ?

```
sed '/\/\*/,\/*\\ / s./\+/' program.c
```



Διαγραφή

d	όλες τις γραμμές
6d	τη γραμμή 6
/^\$/d	όλες τις κενές γραμμές
/^\./d	όλες τις γραμμές που ξεκινούν με .
1,10d	τις γραμμές 1-10
1,/^\$/d	από τη 1η γραμμή μέχρι και την πρώτη κενή γραμμή
/^\$/ , \$d	από την πρώτη κενή γραμμή μέχρι και τη τελευταία γραμμή

```
sed -e '/\/\*/,\/*\\ / s./\+/' -e 's/[ \t]\+/' -e '/$/ d' file.c
```



sed -- απλά παραδείγματα

- ▶ Αντικείμεση το "foo" με το "bar" μόνο στις γραμμές που περιέχουν το "baz"

```
sed '/baz/s/foo/bar/g'
```

- ▶ Σβήσε τα κενά από την αρχή και από το τέλος κάθε γραμμής

```
sed 's/^[ \t]*//;s/[ \t]*$//'
```

- ▶ Πρόσθεσε 5 κενά στην αρχή κάθε γραμμής

```
sed 's/^/     /'
```

- ▶ Σβήσε όλες τις κενές γραμμές από ένα αρχείο (όπως η "grep -v")

```
sed '/^$/d'
```



Γενικά για την awk

awk: Alfred Aho, Peter Weinberger, and Brian Kernighan

- ▶ mini Γλώσσα προγραμματισμού σχεδιασμένη να βρίσκει και να ταιριάζει πρότυπα και να εκτελεί εντολές σε αυτά → Perl
- ▶ Μπορεί να επενεργήσει επί των δεδομένων εισόδου της με σύνθετες, αναδρομικές συναρτήσεις (όπως στη C)
- ▶ Επεξεργάζεται αρχεία κειμένου ανά **πεδία**
- ▶ Τον χρησιμοποιούμε περισσότερο όταν η γνώση για την πληροφορία βρίσκεται στη δομή του αρχείου (γραμμές, στήλες)
- ▶ για πιο σύνθετες πράξεις, όπως αναδρομική εργασία σε συμβολοσειρές ή τη μορφοποίηση εξόδου
- ▶ όταν είναι απαραίτητες ευκολίες μιας υψηλού επιπέδου C-like γλώσσας, όπως πίνακες και εντολές ελέγχου ροής



Χαρακτηριστικά

- ▶ Σε σύνταξη μοιάζει με τη C
- ▶ Χειρίζεται την είσοδο, το διαχωρισμό πεδίων και τη διαχείριση μνήμης αυτόματα
- ▶ Ενσωματωμένοι τύποι δεδομένων για συμβολοσειρές και αριθμούς
- ▶ Όχι δηλώσεις τύπων μεταβλητών
- ▶ Κατάλληλη αριθμητική επεξεργασία
- ▶ Μεταβλητές και ροή ελέγχου στις ενέργειες
- ▶ Βολικός τρόπος πρόσβασης πεδίων μέσα σε γραμμές
- ▶ Εύκαμπτη εκτύπωση
- ▶ Ενσωματωμένες συναρτήσεις αρίθμησης και συμβολοσειρών



Δομή

Ένα πρόγραμμα awk αποτελείται από:

- ▶ Ένα προεραπικό βήμα που εκτελείται πριν το διάβασμα εισόδου BEGIN
- ▶ Ένα ζευγάρι πρότυπο – ενέργεια (pattern – action)
 - ▶ επεξεργάζεται τα δεδομένα εισόδου
 - ▶ εφαρμόζει την εντολή action για κάθε matching
- ▶ Ένα προεραπικό βήμα που εκτελείται μετά το διάβασμα εισόδου END



Πεδία και τελεστές

- ▶ Ένα πρόγραμμα awk μπορεί να είναι ιδιαίτερα πολύπλοκο. Σε αυτό το μάθημα θα γνωρίσουμε την awk μέσα από παραδείγματα που αρκούν για να δουλέψετε στην 1η άσκηση.
- ▶ Για να τρέξουμε ένα awk πρόγραμμα μπορούμε να δώσουμε ένα σύντομο program και input files (προεραπικά, αφού τα δεδομένα μπορεί να έρχονται από pipe ή redirection) ως ορίσματα γραμμής εντολής
- ▶ `awk 'program' [input_file(s)]`
- ▶ Ή στη θέση του 'program' να δώσουμε ένα αρχείο με τις εντολές που αυτό θα περιέχει:
- ▶ `awk -f program_file input_files`



Παραδείγμα 1

Εκτύπωσε όλη την είσοδο τυπώνοντας πριν ένα START και μετά ένα STOP

```
awk 'BEGIN {print "START"} {print} END {print "STOP"}' foo.txt
```

Εκτύπωσε μου τα ονόματα των αρχείων στον τρέκιν κατάλογο και τους χρήστες τους

```
ls -l | awk '
BEGIN { print "File\tOwner" }
{ print $8, "\t",
$3}
END { print "done" }
'
```

Προσέξτε τα αιτιάκια! Αν τα ξεχάσετε, το πρόγραμμα σας δε θα τρέξει!



Παραδείγμα 2

```
bash-3.1$ cat example.awk
```

```
Line number 1
Line number 2
Line number 3
Line number 4
```

```
bash-3.1$ awk '/2/ {print $0} BEGIN {print "Hello
you"} END {print "goudbye"}' example.awk
```

```
Hello you
Line number 2
goudbye
```



Παραδείγμα 3

Εκτύπωσε τις γραμμές των οποίων τα πεδία είναι > 3 και καλύπτουν το πρότυπο `/file/` αλλιώς εκτύπωσε τις γραμμές των οποίων τα πεδία είναι ≤ 3 και ξεκινούν με το `and`

```
bash-3.1$ cat example2.awk
```

```
Just a text file. Nothing to see here.
```

```
Some lines have
more fields than others
```

```
and some
( <- blank line)
```

```
are blank.
```

```
bash-3.1$ awk 'NF>3 ? /file/ : /^and/ {print $0}' \
example2.awk
```

```
Just a text file. Nothing to see here.
and some
```



Παραδείγμα 4

Εκτύπωσε όλα τα αρχεία μη μηδενικού μεγέθους με κατάληξη `.txt`

```
bash-3.1$ ls -l | awk 'BEGIN {print "List of all .txt
files"} /\.txt$/ && $5>0 {print "line number:" NR,
```

```
"file", $9, "of size:", $5} END {print "OK!!!"}'
```

```
List of all .txt files
```

```
line number:9 file file+1.txt of size: 0
line number:12 file output.txt of size: 4898
line number:13 file processes.txt of size: 12953
line number:16 file test-cut.txt of size: 55
line number:19 file test-sort.txt of size: 124
line number:20 file test-tr.txt of size: 40
OK!!
```



awk -- απλά παραδείγματα (1)

- ▶ Τύπωσε τις δύο πρώτες στήλες (default FS=" ") με αντίστροφη σειρά

```
awk '{ print $2, $1 }' file
```

- ▶ Τύπωσε τη στήλη 3 αν η στήλη 1 είναι μεγαλύτερη από τη στήλη 2

```
awk '$1 > $2 {print $3}' file
```

- ▶ Τύπωσε τη γραμμή εμφάνισης, τον αριθμό των στηλών και τη στήλη 1 κάθε γραμμής

```
awk '{print NR, NF, $1}' file
```



awk -- απλά παραδείγματα (2)

- ▶ Τύπωσε το αρχείο file αφού διαγράψεις τη 2η στήλη του

```
awk '{ $2 = ""; print }' file
```

- ▶ Τύπωσε όλες τις γραμμές που διαφέρουν ανά δύο στην πρώτη στήλη

```
awk '$1 != prev { print; prev = $1 }' file
```

- ▶ Τύπωσε μόνο την τελευταία γραμμή

```
awk '{ line = $0 } END { print line }'
```



Τι είδαμε από την awk?

- ▶ Υπάρχουν κάποιες ενδιαφέρουσες μεταβλητές περιβάλλοντος της γλώσσας, όπως οι FS (Field Separator), NF (Number of fields), NR (Number of Records read so far)
- ▶ Μπορούμε να ξεχωρίζουμε τα πεδία (στήλες) της εισόδου αναφερόμενοι σε αυτά ως \$ και τον αριθμό τους
- ▶ Μπορούμε να επιλέξουμε, με patterns, τις γραμμές της εισόδου επί των οποίων θα εφαρμοστεί το πρόγραμμα μας. Μάλιστα αυτά τα patterns μπορεί να έχουν και κάποια λογική



awk - μεταβλητές, ροή ελέγχου

Λειτουργίες με μεταβλητές:

- ▶ Βρες το άθροισμα και τον μέσο όρο των αριθμών της 1ης στήλης

```
awk '{ s += $1 } END { print "sum is", s, " average is", s/NR }'
```

- ▶ **Ροή ελέγχου: for loop**

Τύπωσε όλες τις στήλες με αντίθετη σειρά

```
awk '{ for (i = NF; i > 0; --i) print $i }' file
```

- ▶ **Ροή ελέγχου: if-else**

Τύπωσε όλες τις μη κενές γραμμές των οποίων οι δυο πρώτες στήλες είναι ίσες

```
awk 'NF > 0 { if ($1 == $2) print $0 }' file
```



awk - μορφοποίηση εξόδου και συνδυασμοί

- ▶ **Μορφοποίηση με printf**

Τύπωσε και στοίχισε όλα τα αρχεία 1000 bytes και τα μεγέθη τους

```
ls -l | awk '{ if ($5 < 1000) printf ("File: %10s - Size %5d\n", $9, $5) }'
```

- ▶ **Συνδυαστικό παράδειγμα**

Πρόσθεσε τους αριθμούς από τις πρώτες μισές γραμμές του file

```
awk '{ while (counter <= NR/2) { for (i=1; i <= NF; i++) sum += $i counter++ } } END { printf("sum of first %2d lines: %3d\n", counter, sum) }'
```

