



Διάλεξη 9 Προγραμματισμός Κελύφους

Δημήτρης Ζεϊναλιπούρ



Περιεχόμενο Διάλεξης

Προγραμματισμός Κελύφους

- Συνθήκες Ελέγχου (If, Case)
- Λογικοί Τελεστές (&&, ||, !)
- Σχεσιακοί Τελεστές (-gt, -lt, ...)
- Αριθμητικές Εκφράσεις και Πίνακες
- Επαναληπτικοί και Ένθετοι Βρόχοι
- Έλεγχοι Αρχείων (File Testing)
- Συναρτήσεις
- Πίνακες και Παραδείγματα





Συνθήκη Ελέγχου if

- Σύνταξη

```
if [ CONDITION1 ]; then  
    if [ CONDITION11 ]; then  
        .....  
    fi  
elif [ CONDITION2 ] ; then  
    STATEMENTS2  
else  
    STATEMENTS3  
fi
```



Η εντολή *test*

Σύνταξη: *test* expression [expression]

Σκοπός: Όπως και το [], να αποτιμήσει την έκφραση 'expression' και να επιστρέψει true ή false

Παράδειγμα:

```
if test -w "$1"; then  
    echo "The file $1 is write-able"  
fi
```

...είναι το ίδιο με το

```
if [ -w $1 ]; then  
    echo "The file $1 is write-able"  
fi
```



Παράδειγμα: File Testing

```
#!/bin/bash
echo -n "Enter a filename: "
read filename
if [ ! -r "$filename" ]; then
    echo "File is not read-able"
    exit 1
fi
```

Αποφεύγει το newline στο τέλος του prompt

Εκτύπωση Return Value Προηγούμενης Εντολής με \$?

```
test "abc" = "def";
```

echo \$? (δηλ., το test έκανε exit 1)

1 # **FALSE** (in the C language, 1 == TRUE)

```
[ "abc" != "def" ];
```

echo \$? (δηλ., το test έκανε exit 0)

0 # **TRUE** (in the C language, 0 == FALSE)



Σχεσιακοί Τελεστές

Επεξήγηση	Αριθμητική	Αλφαριθμητική
Greater than	-gt	s1 > s2
Greater than or equal	-ge	s1>s2 s1=s2
Less than	-lt	s1 < s2
Less than or equal	-le	s1<s2 s1=s2
Equal	-eg	=
Not equal	-ne	!=
String length is greater than zero	[-n "abc"]; echo \$? 0 #TRUE	-n str
String length is zero	[-z ""]; echo \$? 0 #TRUE	-z str

Δήλωση Ακεραίου
declare -i var



File Testing

Έκφραση

Τιμή Επιστροφής

-d file

True if 'file' is a **directory**

-f file

True if 'file' **exists on disk**

-r file

True if 'file' is **readable**

-w file

True if 'file' is **writable**

-x file

True if 'file' is **executable**

-s file

True if **length of 'file' is nonzero**



Παράδειγμα: File Testing

```
#!/bin/bash
```

```
#είναι καλό τα ονόματα αρχείων να δηλώνονται στην αρχή του script
```

```
TMPFILE="diff.out"
```

```
diff $1 $2 > $TMPFILE
```

-s: Ελέγχει εάν το αρχείο είναι μη-κενό

```
if [ ! -s "$TMPFILE" ]; then # δηλ. είναι κενό το $TMPFILE
```

```
    echo "Files are the same!"
```

```
else
```

```
    more $TMPFILE
```

-s: Ελέγχει εάν το αρχείο υπάρχει στον δίσκο

```
fi
```

```
if [ -f $TMPFILE ]; then
```

```
    rm -rf $TMPFILE
```

```
fi
```




Παράμετροι Επιλογής -parameter

Το πιο κάτω παράδειγμα μας επιτρέπει να χειριστούμε τις τρεις περιπτώσεις:

- A) Χωρίς Παραμέτρους: π.χ., *./command.sh*
- B) Με Παράμετρο Επιλογής *-c* π.χ., *./command.sh -c*
- C) Με Παράμετρο, αλλά όχι Επιλογής π.χ., *./command.sh costas maria christos*

```
#!/bin/bash                                1,2,...
if [ $# -eq 0 ]; then # No arguments
    echo "There are no arguments!"
elif [ x$1 = x-c ]; then # Ελέγχει εάν δόθηκε το -c
    echo "Command Line Option -c"
else
    echo "$# parameters: $" #Επιστρέφει 3 και μια
    λίστα με τις παραμέτρους $1 $2 $3 ...
fi
```



Λογικοί Τελεστές

- Μας επιτρέπουν να δημιουργήσουμε **σύνθετες λογικές εκφράσεις (compound statements)**.
- Θα δούμε τρεις τελεστές
 - **AND (&&)**
 - **OR (||)**
 - **NOT (!)**
- **If `[[State1 && State2]]`; then**
Εκτέλεσε το State1, και εάν το exit status είναι TRUE, τότε εκτέλεσε και το State2.

*Η διπλή αγκύλη `[[]]` : Ο νέα έκδοση της `TEST []`. Υποστηρίζεται από το `BASH` αλλά όχι από παλιότερα κελύφη. Επιτρέπει πιο αναγνώσιμες εκφράσεις (π.χ., χρήση `&&`, `||`, κτλ. αντί `-a`, `-o`. Επίσης το `[[]]` επιτρέπει *pattern matching* π.χ., `[[$file = r*]] && echo "$file starts with r name"`*



Παράδειγμα Άρνησης !

```
#!/bin/bash
```

```
read -p "Enter years of service: " Years
if [ ! "$Years" -lt 50 ]; then # (Years>=50)
    echo "You can retire now."
else
    echo "You can' t retire."
fi
```



Παράδειγμα Διάζευξης II

```
#!/bin/bash   Δώσε Bonus στους Παραγωγικούς || Ευγενικούς Τηλεφωνητές!  
read -p "Enter the calls handled: " CallsHandled  
read -p "Enter the calls closed: " CallsClosed  
if [ $CallsHandled -gt 150 ] || [ $CallsClosed -lt 10 ];  
    then  
        echo "You are entitled to a bonus!"  
else  
    echo "You are only entitled to a bonus if the calls"  
    echo "handled exceeds 150 or calls closed is less"  
    echo "than 10."  
fi
```



Παράδειγμα Σύζευξης &&

```
#!/bin/bash
Bonus=500
read -p "Enter Status: " Status
read -p "Enter Shift: " Shift
if [[ $Status = "hourly" && $Shift -eq 3 ]]; then
    echo "Your bonus if working shift $Shift is
    \b$Bonus."
else
    echo "You are only entitled to a bonus if you "
    echo "are hourly and work shift 3."
fi
```

*Δώσε Bonus=500 στους ωριαίους
υπαλλήλους της βάρδιας 3*

Παραδείγματα File Testing Και Λογικών Τελεστών



`#!/bin/bash` Αριθμός Command Line Arguments

`if [$# -lt 1]; then`

`echo "Usage: $0 filename"`

`exit 1`

`fi`

`if [! -w $1]; then`

`echo "File $1 is not writeable"`

`exit 1`

`fi`



Αριθμητικές Εκφράσεις

- Οι αριθμητικές εκφράσεις (**μόνο ακέραιοι!**) αποτιμώνται με τον ακόλουθο τρόπο:

`((a=a+1))` ή **`a=$((a+1))`** ή **`a=$(($a+1))`**
`let a=a+1` ή **`let a++`** (απλοποιημένη *expr*) ή
`a=`expr $a + 1``

- Οι τελεστές είναι περίπου οι ίδιοι με τη γλώσσα C (δες επόμενη διαφάνεια)
- Μια μεταβλητή **δεν χρειάζεται** να είναι δηλωμένη (`declare -i a`) σαν integer για να χρησιμοποιηθεί στις μαθηματικές εκφράσεις.
- ΔΕΝ επιτρέπονται οι Floating Point Πράξεις στο Bash ... αλλά υπάρχει τρόπος μέσω της εντολής “bc”
 - <http://www.linuxjournal.com/content/floating-point-math-bash> 9-15



Αριθμητικές Πράξεις

Operator	Meaning
VAR++ and VAR--	variable post-increment and post-decrement
++VAR and --VAR	variable pre-increment and pre-decrement
- and +	unary minus and plus
! and ~	logical and bitwise negation
** ((a=2**3)) echo \$a # δίνει 8	exponentiation
*, / and %	multiplication, division, remainder
+ and -	addition, subtraction
<< and >> ((a=8>>1)); echo \$a #δίνει 4	left and right bitwise shifts
<=, >=, < and >	comparison operators
== and !=	equality and inequality
&	bitwise AND
^	bitwise exclusive OR
	bitwise OR
&&	logical AND
	logical OR
expr ? expr : expr	conditional evaluation
=, *=, /=, %=, +=, -=, <<=, >>=, &=, ^= and =	assignments
, ((a=a+1,b=b+1))	separator between expressions

Κάποια ισχύουν μόνο για την έκφραση ((a=a+1)) και όχι για την **let** ρ ©

Παράδειγμα με Αριθμητικές Πράξεις



```
$ more Income
```

```
#!/bin/bash
```

```
read -p "Enter Income Amount: " Income
```

```
read -p "Enter Expenses Amount: " Expense
```

```
((Net=Income-Expense))
```

```
# ή εναλλακτικά : let "Net=$Income - $Expense"
```

```
if [ "$Net" -eq "0" ]; then
```

```
    echo "Income and Expenses are equal - breakeven."
```

```
elif [ "$Net" -gt "0" ]; then
```

```
    echo "Profit of: " $Net
```

```
else
```

```
    echo "Loss of: " $Net
```

```
fi
```



Η εντολή case

- Μας παρέχει ένα μηχανισμό για **multi-way branching** αντίστοιχο της **switch(case)** στην C,

```
case STRING is  
pattern)
```

```
    STATEMENTS1 ;;
```

```
pattern|pattern)
```

```
    STATEMENTS2 ;;
```

```
*) echo "Invalid choice!"; exit 1 ;;
```

```
esac
```

- Αντιθέτως με την C, δεν υπάρχει κάποιος περιορισμός ως προς τον **τύπο της μεταβλητής** στο pattern (ενώ στη c πρέπει να είναι scalar)

Παράδειγμα 1 με την case



Ένα τροποποιημένο ls command

#!/bin/bash *Ότι κείμενο υπάρχει σε αυτό το block θα αναγραφεί στο terminal - Αυτό ονομάζεται Here-Document*

cat << END

Enter Y to see all files w/ hidden files.

Enter N to see all non-hidden files.

Enter q to quit.

END

read reply

<<< denotes a here **string**.

```
# echo "hi there" | hexdump -C
hexdump -C <<< 'hi there'
00000000  68 69 20 74 68 65 72 65  0a
|hi there.|
00000009
```

<< denotes a here **document**.

The translate command (tr), converts lower case letters to upper case

REPLY=`echo \$reply | tr [:lower:] [:upper:]`

case \$REPLY in

Y|YES) echo "Displaying all files including hidden files..."

ls -a ;;

N|NO) echo "Display all non-hidden files..."

ls ;;

Q) exit 0 ;;

***)** echo "Invalid choice!"; exit 1 ;;

esac

Here Document με XML parsing



```
cat >AdditionalLanguages.plist << EOF
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Xcode.SourceCodeLanguage.BibTeX</key>
    <dict>
        <key>languageSpecification</key>
        <string>tex.lang.bibtex</string>
        <key>fileDataType</key>
    </dict>
</dict>
</plist>
```

```
EOF
```

```
cat AdditionalLanguages.plist | <do-some-xml-parsing>
```

Some Ideas:

⇒ *sgrep* (structured grep) is a tool for searching and indexing text, SGML, XML and HTML files and filtering text streams using structural criteria

⇒ *xmlint* — command line XML tool

⇒ **Xmllsh** - A command line shell for XML

Παράδειγμα 2 με την case



Εύρεση του Περιεχομένου και του Extension κάποιου Αρχείου

```
#!/bin/bash
if [ ! -f $1 ]; then
    echo "Usage: $0 valid-filename";
    exit 1
fi
```

```
$ file exercise.pdf
exercise.pdf: PDF document, version 1.2
```

```
Filetype=`file $1 | awk -F":" '{print $2}` # file: βρίσκει το είδος του αρχείου
echo "Based on the contents, this is a:"
echo $Filetype
```

```
echo -n "Extension: "
case $1 in
    *.c) echo "C Program";;
    *.cpp) echo "C++ Program";;
    *.java) echo "Java Program";;
    *.class) echo "Java Bytecode";;
    *.sh) echo "Shell Script";;
    *) echo "Unknown";;
```

```
$ ./test.sh a.sh
```

```
Based on the contents, this is a:
Bourne-Again shell script text executable
Extension: Shell Script
```



Επαναληπτικός Βρόχος while

Σύνταξη:

```
while [ expression ];  
do  
    command-list  
done
```

Εκτύπωση 0 έως 9

```
i=0  
while [[ $i -lt 10 ]];  
do  
    echo $i  
    ((i++))  
done
```

**Άπειρος Βρόχος με
περιοδικά μηνύματα**
while **true**;

```
do  
    echo "Still Alive"  
    sleep 3 # seconds  
done
```

Παράδειγμα Χρήσης του *until*



```
#!/bin/bash
```

```
Stop="N"
```

```
until [[ $Stop = "Y" ]]; do
```

```
    ps -ef
```

```
    read -p "Do you want to stop? (Y/N)" reply
```

```
    Stop=`echo $reply | tr [:lower:] [:upper:]`
```

```
done
```

```
echo "Stopping..."
```

```
./execut.sh
```

	UID	PID	PPID	TTY	STIME	COMMAND
dzeina	3180	1	con	23:51:11	/usr/bin/bash	
dzeina	392	3180	con	23:53:42	/usr/bin/bash	
dzeina	3384	392	con	23:53:42	/usr/bin/ps	

Do you want to stop? (Y/N)Y
Stopping...



Η εντολή *break*

Έξοδος από επαναληπτικό βρόχο με την εντολή **break**, όπως και στην C

```
while condition
```

```
do
```

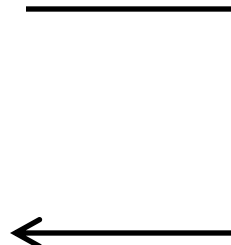
```
    cmd-1
```

```
    break
```

```
    cmd-n
```

```
done
```

```
echo "done"
```






Η εντολή *continue*

Περισσότερος Έλεγχος Ροής Προγράμματος,
όπως και στην C.

```
while [ condition ]  
do  
    cmd-1  
    continue  
    cmd-n  
done  
echo "done"
```





Ο Επαναληπτικός Βρόχος *for*

Σύνταξη:

```
for variable in argument-list
do
    cmd-list
done
```

- Χωρίς το «*in argument-list*», η επανάληψη θα εκτελεστεί για το **\$***, το οποίο απεικονίζει τις παραμέτρους εισόδου μιας εντολής (*command line parameters*).



for Loop: Παράδειγμα

```
#!/bin/bash  
for i in 7 9 2 3 4 5  
do  
    echo $i  
done | sort -n
```

Επιστρέφει

\$

2

3

4

5

7

9

 **sort numerically**

for Loop: Παράδειγμα



Εύρεση της μέσης θερμοκρασίας

```
#!/bin/bash
Days=7
for num in 1 2 3 4 5 6 7
do
    read -p "Enter Temp for day $num: " Temp
    ((TempTotal=TempTotal+Temp))
done

((AvgTemp=TempTotal / Days))
echo "Average temp was: " $AvgTemp
```

for Loop: Παράδειγμα



Εύρεση των χρηστών του συστήματος (εντολή users)

```
#!/bin/bash
```

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

Εκτύπωση όλων των αρχείων C

```
do
```

```
    echo $i
```

```
done
```

```
# Εκτύπωση όλων των χρηστών
```

```
for i in `ps -ef | awk '{print $1}' | sort | uniq`
```

```
do
```

```
    echo $i
```

```
done
```

Στιγμιότυπο της “ps -ef”

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	13952	1	0	Jan17 ?		00:00:00	[lockd]
root	19559	1	0	04:17 ?		00:00:00	cupsd
root	22733	2212	0	16:01 ?		00:00:00	sshd: dzeina [priv]
dzeina	22735	22733	0	16:01 ?		00:00:00	sshd: dzeina@pts/3
dzeina	22736	22735	0	16:01	pts/3	00:00:00	-bash
dzeina	22837	22736	0	16:15	pts/3	00:00:00	ps -ef



Συναρτήσεις Κελύφους

- **Ώρα για πιο δομημένο προγραμματισμό!**
- Οι συναρτήσεις στο κέλυφος, μπορούν να πάρουν διάφορες παραμέτρους και να επιστρέψουν κάποια τιμή εξόδου.
- Ο ορισμός των συναρτήσεων **πρέπει** να γίνεται στην αρχή του script, προτού καλέσουμε την συνάρτηση, για να μπορεί ο μεταφραστής να γνωρίζει για την ύπαρξη της συνάρτησης
- Η εκτέλεση των συναρτήσεων και εντολών εξακολουθεί να είναι ακολουθιακή



Συναρτήσεις Κελύφους

Σύνταξη:

Προαιρετικό

```
function function-name [( )]  
{  
    statements  
    [return]  
}
```



Παράδειγμα Συνάρτησης

```
#!/bin/bash
```

```
# Ορισμός Συνάρτησης
```

```
fun () {
```

```
    i=0
```

```
    REPEATS=5
```

```
    while [ $i -lt $REPEATS ]
```

```
    do
```

```
        echo "Still Alive"
```

```
        sleep 1
```

```
        ((i++))
```

```
    done
```

```
}
```

```
# Κλήση Συνάρτησης
```

```
fun
```




Συναρτήσεις με Παραμέτρους

- Μια συνάρτηση μπορεί, όπως και ένα πρόγραμμα κελύφους, να λάβει παραμέτρους (\$1, \$2,...).

```
#!/bin/bash
```

```
user_greet ()  
{  
    echo "Hello, $1."  
}
```

```
user_greet $USER
```



Εμβέλεια Μεταβλητών Συναρτήσεων

- Οι μεταβλητές που ορίζονται μέσα στις συναρτήσεις είναι εξορισμού **καθολικές** (δηλ., η τιμή τους παραμένει καθ'ολη την διάρκεια ενός προγράμματος κελύφους).
- Αυτό μπορεί να οδηγήσει σε **διαφορά σφάλματα** εάν τα προγράμματα είναι **μεγάλα** και χρησιμοποιούν **κοινότυπα** ονόματα.
- Η λέξη **“local”** μέσα σε μια συνάρτηση περιορίζει την εμβέλεια μιας μεταβλητής μέσα στην συνάρτηση στην οποία ορίζεται (δηλαδή αυτό που συμβαίνει εξ' ορισμού στην C)
 - Μπορείτε και με το «**declare**» να περιορίσετε την εμβέλεια μέσα στην συνάρτηση (π.χ., `declare -i someinteger`)
 - When used in a function, **declare** makes each *name* **local**, as with the `local` command, unless the **-g** option is used"



Example: function

```
#!/bin/bash

global="pretty good variable"

foo () {
    local inside="this is a local variable"
    echo $global
    echo $inside
    global="better variable"
}
```

```
echo $global
foo
echo $global
```

Output:

```
pretty good variable
pretty good variable
this is a local variable
better variable
```

```
echo $inside #nothing printed here!
```

Πίνακες



- Μια μεταβλητή μπορεί να χρησιμοποιηθεί σαν πίνακας, εάν ακολουθείται από την έκφραση [index],
- **Δήλωση Τύπου Πίνακα (Προαιρετικό)**
declare -a ARRAY
- Αναφορά σε κάποιο στοιχείο γίνεται με τον συνηθισμένο τρόπο array
ARRAY[0]=2;
echo \${ARRAY[0]}
- Το μέγεθος του πίνακα δεν δηλώνεται όπως άλλες γλώσσες

Πίνακες: Αρχικοποίηση & Χρήση



```
#!/bin/bash
```

```
# ή declare (δήλωση ότι πρόκειται για πίνακα)
```

```
set -a ARRAY
```

```
ARRAY[0]="zero"
```

```
ARRAY[1]="one"
```

```
# σε ένα πίνακα μπορούμε να αναθέσουμε
```

```
# οποίο τύπο δεδομένων θέλουμε...είναι όλα Strings...
```

```
ARRAY[2]=2.0
```

```
ARRAY[3]=3
```

```
ARRAY[4]="four"
```

```
echo ${ARRAY[*]}
```

```
>> Εκτυπώνει zero one 2.0 3 four'
```

Πίνακες: Αρχικοποίηση & Χρήση



Συνέχεια...

three=3

echo \${ARRAY[three]} # Εκτυπώνει 3

Μέγεθος του Πίνακα (δηλ. 5)

\${#ARRAY[*]}

διαγράφει τα στοιχεία του πίνακα

unset ARRAY

Εναλλακτικός Ορισμός Πίνακα

ARRAY=(1 2 3 4 5)

Πίνακες: Αρχικοποίηση & Χρήση

Παράδειγμα: Δημιουργία Πίνακα 1000 Ακεραίων



```
#!/bin/bash
declare -i i=0
declare -a ARRAY

while [ $i -lt 1000 ];
do
    ARRAY[$i]=$i # assign $i to position $i
    ((i++))
done
echo ${ARRAY[*]}; # print the ARRAY
```

Μελέτη Προβλήματος I



Σας δίδεται μια λίστα A με τα ονόματα κάποιων αρχείων και καταλόγων.

Γράψετε ένα πρόγραμμα σε κέλυφος Bash το οποίο **εκτυπώνει στην οθόνη** τα αρχεία και καταλόγους τα οποία δεν υπάρχουν στο δίσκο.

Μελέτη Προβλήματος



```
#!/bin/bash
```

```
for i in `cat list.txt`;  
do
```

```
    if [[ ! -f $i && ! -d $i ]]; then
```

```
        echo $i does not exist
```

```
    fi
```

```
done
```

Μελέτη Προβλήματος II



Δημιουργήστε ένα menu μέσω του
οποίο θα δίδετε στον χρήστη την
δυνατότητα να επιλέξει ανάμεσα στις
ακόλουθες επιλογές

Menu

- 1 show the date
- 2 show the current dir
- 3 list the current dir
- 4 edit a file
- 5 backup .profile to /tmp/backup.\$USER

Μελέτη Προβλήματος



echo "

Menu

- 1 show the date
- 2 show the current dir
- 3 list the current dir
- 4 edit a file
- 5 backup the system

Enter your choice: \c"

read CHOICE

Μελέτη Προβλήματος



```
case "$CHOICE" in
  1) date ;;
  2) pwd ;;
  3) ls ;;
  4) echo "Enter filename to edit: \c"
     read FILE
     if [ ! -r "$FILE" ]
     then
       echo "Error: cannot access $FILE" >&2
       exit 1
     fi
     vi $FILE
     ;;
  5) systembackup;;
  *) echo "Invalid option" >&2 ;;
```

Μελέτη Προβλήματος



```
systembackup() {  
  LOG=/tmp/backup-$USER.log  
  TAPE=/tmp/backup-$USER.bak  
  (  
    echo "Are you sure you want to backup now? \c" > /dev/tty # αυτό γραφεί στο terminal  
    read ANS  
    case "$ANS" in  
      [Nn]*) exit;;          # Δέχεται και το No nO NO!  
      [Yy]*) ;;              # just fall through if yes  
      *)  
        echo "Invalid input. Program aborted" > /dev/tty # αυτό γραφεί στο terminal  
        # 1st echo is for the screen  
        echo "Invalid input. Program aborted `date`" # αυτό γραφει στο $LOG  
        # 2nd echo is for log  
        exit 1 ;;  
    esac  
    echo "start backup `date`"  
    cat ~/.profile > $TAPE # Εδώ γίνεται το backup (μονάχα του .profile)  
    echo "end backup `date`"  
  ) >> $LOG 2>&1 # τα echo στη παρένθεση γίνονται append στο LOG (εφόσον το STDERR και  
  # STDOUT ανακατευθύνονται στο LOG)  
}
```

Πρόγραμμα Συλλογής Κατανεμημένων Χρηστών



```
#!/bin/bash
COMMAND="ps -ef"
echo "Running $COMMAND"
for i in `cat hostnames.txt`
do
    # echo -n " $i"
    # assuming public/private key has been established
    ssh $i "$COMMAND > /tmp/file " &
    # echo "...Done"
done

echo "Waiting"
sleep 1

echo "Collecting Data"
for i in `cat hostnames.txt`
do
    # echo -n " $i"
    ssh $i "cat /tmp/file " &
    #echo "...Done"
done | awk -F" " '{print $1}' | sort | uniq
```

cat hostnames.txt

```
b103ws1.in.cs.ucy.ac.cy
b103ws2.in.cs.ucy.ac.cy
b103ws3.in.cs.ucy.ac.cy
b103ws4.in.cs.ucy.ac.cy
b103ws5.in.cs.ucy.ac.cy
b103ws6.in.cs.ucy.ac.cy
b103ws7.in.cs.ucy.ac.cy
b103ws8.in.cs.ucy.ac.cy
b103ws9.in.cs.ucy.ac.cy
b103ws10.in.cs.ucy.ac.cy
b103ws11.in.cs.ucy.ac.cy
b103ws12.in.cs.ucy.ac.cy
b103ws13.in.cs.ucy.ac.cy
b103ws14.in.cs.ucy.ac.cy
b103ws15.in.cs.ucy.ac.cy
b103ws16.in.cs.ucy.ac.cy
```