Linux Shell

:~>touch اسم الملف.txt إنشاء ملف

==========

:~>cat اسم الملف لاستعراض الملف

:~>cat qasem.txt qasem.txt يستعرض الملف

:~>cat -n اسم الملف لاستعراض الملف مع ترقيم الأسطر

==========

:~>mkdir اسم المجلد إنشاء مجلد جديد

:~>mkdir qasem إنشاء المجلد qasem

:~>mkdir r1/r2 (error if r1 not found)

:~>mkdir .q إنشاء المجلد q وجعله مخفي

==========

:~>cd اسم المسار الإنتقال الى المسارات

:~>cd qasem الإنتقال الى داخل المجلد qasem

:~>cd الإنتقال الى الجذر

==========

نقل ملف من مكان لآخر الاسم٢/المكان٢ الاسم١/المكان١ mv>~:

ننشئ المجلدات d1/d2 و d3/d4 ونكتب الأوامر التالية:

d3/d4/ d1/d2 mv>~: نقل d2 من d1 إلى d4

d3/d4 d1/d2 mv>~: نقل d2 من d1 إلى d3 الذي يحوي d4 وبالتالي :

أحيانا يخبرنا بأنه موجود وأحيانا يستبدله (في الملفات يستبدل دوما)

~/d3/d4/d7 ~/d1/d2 mv>~ : نقل d2 من d1 إلى d4 وإعادة تسميتها إلى d7

ملاحظة هامة: لا يمكن نقل ملف إلى مكان غير موجود

==========

انشاء اختصار ln

انشاء اختصار لـ الاسم١ اسم الإختصار/المكان٢ الاسم١/المكان١ ln>~:

- حذف الملف الأصلي -> الاختصار لا يعمل
- حذف link لا يحصل شيء

(ولكن أي عمليات نجريها على link تتم على الملف الأصلي)

==========

بالنسبة للملفات cp

نسخ ملف من مكان لآخر مع إعادة التسمية الاسم٢/المكان٢ الاسم١/المكان١ cp>~:

بالنسبة للمجلدات cp

نسخ مجلد من مكان لآخر /المكان٢ اسم المجلد١/المكان١ cp>~:

نسخ مجلد من مكان لآخر مع استبداله اذا كان موجود /المكان٢ اسم المجلد١/المكان١ cp>~:

نسخ مجلد من مكان لآخر مع إعادة التسمية الاسم٢/المكان٢ الاسم١/المكان١ cp>~:

==========

==========

حذف ملف :~>rm اسم الملف

حذف الملف qasem.txt :~>rm qasem.txt

==========

حذف مجلد :~>rmdir اسم المجلد

حذف المجلد qasem :~>rmdir qasem

(شرط أن يكون المجلد فارغ)

حذف المجلد q المخفي :~>rmdir .q

حذف المجلد xxx الممتلئ مع السؤال :~>rm -r xxx

حذف المجلد xxx الممتلئ دون السؤال :~>rm -rf xxx

==========

==========

:~>ls اسم المسار استعراض المسار

:~>ls ~ home directory استعراض المسار

:~>ls -L اسم المسار استعراض المسار بكامل التفاصيل

:~>ls -a اسم المسار استعراض المسار مع اظهار المجلدات المخفية

:~>ls -a اسم المسار استعراض المسار مع اظهار المجلدات المخفية

:~>ls -La اسم المسار استعراض المسار مع اظهار المجلدات المخفية والغير مخفية

:~>ls -R اسم المسار استدعاء ls على جميع المجلدات الفرعية

:~>ls -L اسم المسار استعراض المجلدات بشكل عامودي

:~>ls -h لعرض option

==========

grep print lines matching a word

grep [options] word [FILE...]

grep searches the named input FILEs (or standard input if no files are named, or the file name - is given) for lines containing a match to the given word.

By default, grep prints the matching lines.

==========

==========

man pages:

:~>man اسم الأمر عرض تفاصيل عن الأمر

:~>man ls شرح التعليمة وللخروج نضغط q

الصفحات manual pages مقسمة إلى أجزاء :

Ch1 : command

Ch2 : system calls

Ch3 : library calls

Ch4 : game calls

Ch5 : الملفات التي لها تنسيق محدد

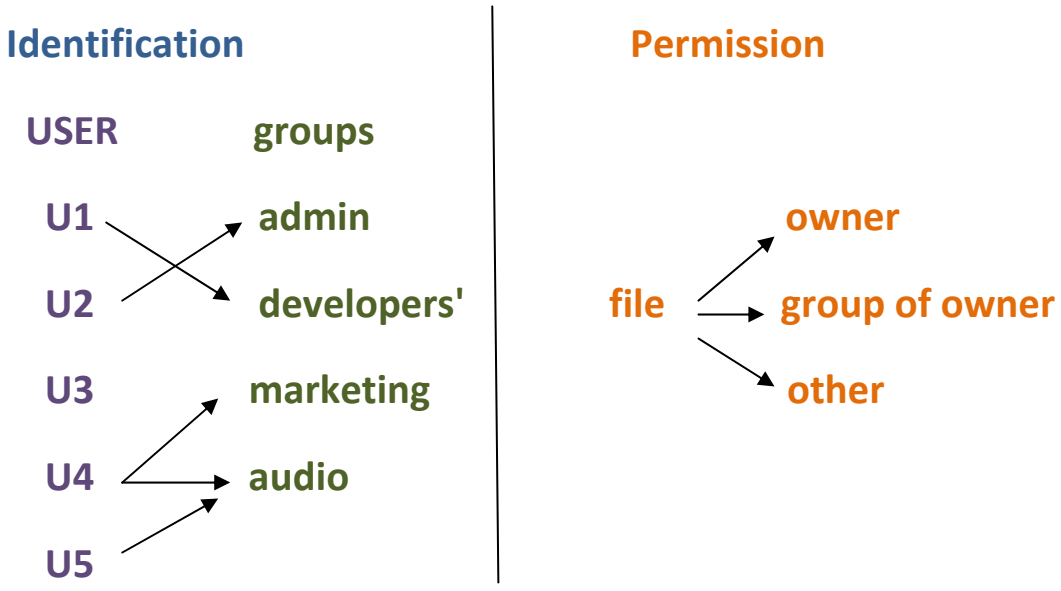
. . . .

Example:

:~>man 2 read

==========

File permission model:



يتم تعريف مجموعة مستخدمين users (multiuser) ونتيجة اشتراك كل مجموعة منهم بعمل معين نتجت group . كل user يجب أن ينتمي لـ group واحدة على الأقل .

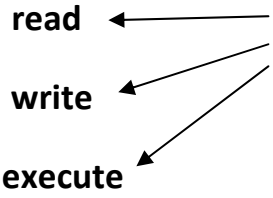
النظام يتعامل مع المستخدمين والمجموعات من خلال أرقام .

وفي linux ← everything is a file وبالتالي عند تطبيق أوامر معينة يتم تطبيقها على النظام ككل .

Permission

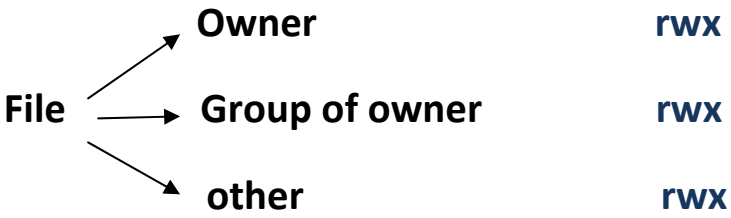
يتم تحديد صفة على الاقل لكل user

العمليات الأساسية على أي file



وهي السماحيات التي يمكن منحها لأي مستخدم أو حجبها .

كل file له 9 خانات للسماحيات + خانة لتحديد نوع الملف



وبالتالي عند عرض خصائص الملف ينتج :

نوع الملف	owner	groupofowner	other
~	- - -	- - -	- - -
–	rwx	r - -	r - -

الـ owner يمكنه عمل read write execute

الـ grop of owner يمكنه عمل read فقط

الـ other يمكنه عمل read فقط

إن الخاصيـــة execute تقابل الدخول إلى مجلد (directory) لكن :

عدم وجود x على directory لا يعني أنني لا يمكن استعراض محتوياته او تعديلها

مثال ١ :

d1/d2/d3/d4/file

إذا كان آخر مجلد لا يملك x أستطيع الوصول للملف والتعديل عليه و . . . .

أي: x تحجب ما يليها وللأسفل ولا تحجب مكانها

مثال ٢:

:~>rmdir d1/d2/d3 مطلوب حذف d3

هل معي write على d2 ← نعم ← يمكن

لا ← لا يمكن

قلنا سابقاً بأن النظام لا يتعامل مع أحرف بل أرقام :

rwx	rwx	rwx
111	110	100
7	6	4

هذا الكود يعبر عن السماحيات السابقة

وزن خانة :

r ← 4 | w ← 2 | x ← 1

إنشاء	تعديل	حذف
:~>useradd	:~>usermod	:~>userdel
:~>groupadd	:~>groupmod	:~>groupdel

:~>chmod تعديل permissions

:~>chown تغيير المالك

:~> chgrp تغيير المجموعة

:~>umask الرقم الجديد تغيير الماسك

السماحيات الإفتراضية :

تحدد صلاحيات إفتراضية لكل مستخدم عن طريق umask :

Directory	7	7	7
File	6	6	6
Umask	0	2	2
Directory	7	5	5
	rwx	– wx	– wx
File	6	4	4
	rw-	r - -	r - -

أي إن الـ full_permission للملف هي 666 أي لا يمكن إنشاء file بطبيعته (بشكل إفتراضي) قابل للتنفيذ (أي لا يحوي x) . إنما يمكن إضافة x لاحقاً بعد الإنشاء .

خطأ فادح :

لنفرض أن الـ umask هـــــو 101 عندها

File	6	6	6
Umask	1	0	1
	1	0	1
خطأ ← x	r	-	

لأن الملف أصلا لا يحوي x والعملية ليست عملية طرح عادية . .

مثال :

إذا أردنا ملف بالسماحيات التالية :

rw -	r - -	- - -
6	4	0

ما هو الـ mask ؟

لأن 026

6	6	6
0	2	6
6	4	0

==========

:~>useradd username -G groupname (حتما المجموعة موجودة)

-d /home/username (by default)

-n (create home directory)

لتحديد الـ shell : تختلف بطريقة - s الدخل والخرج والأوامر وغير ذلك

والإفتراضي هو bash (default) ويمكن تغييرها .

مثال :

لجعل الـ shell الإفتراضية cshell : - s /bin/csh

ميزة الـ bash أنها موجودة على جميع أنظمة Unix .

- /bin/bash

- /bin/false ⟵ حالة خاصة

وذلك من أجل الـ security لأمور تنظيمية .

:~>userdel username حذف مستخدم

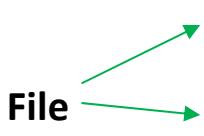
:~>usermod username تعديل مستخدم

فائدة المجموعة : تجميع المستخدمين الذين لهم نفس الوظيفة في نفس المجموعة .

:~>groupadd groupname إضافة مجموعة

:~>groupdel groupname حذف مجموعة

لكل ملف خاصيتين أساسيتين :




```
:~>chown  u2  files
```

المالك الجديد الملفات المراد تغييرها

يحق تنفيذها من قبل : root و مالك الملف owner .

ملاحظة: لا يمكن أن يكون للملف أكثر من مالك ولو قمنا بنسخ الملف فإنه ينتج ملف جديد وليس نفس الملف .

لدينا المثال التالي :

```
:~>chown  u2  dir
```
(يغيير الـ dir فقط)

```
:~>chown –r u2  dir
```
(يغيير الـ dir وكل ما بداخلها)

سؤال: إذا كان داخل الـ dir ملفات لمالكين آخرين ؟

ج : يتم تغيير ملكية الملفات التابعة للمالك فقط .

تغيير المجموعة (تغيير مجموعة الملف) :

```
:~>chgrp  g2  dir
```
تغيير المجموعة الخاصة بالملف

وليس تغيير المجموعة بحد ذاتها

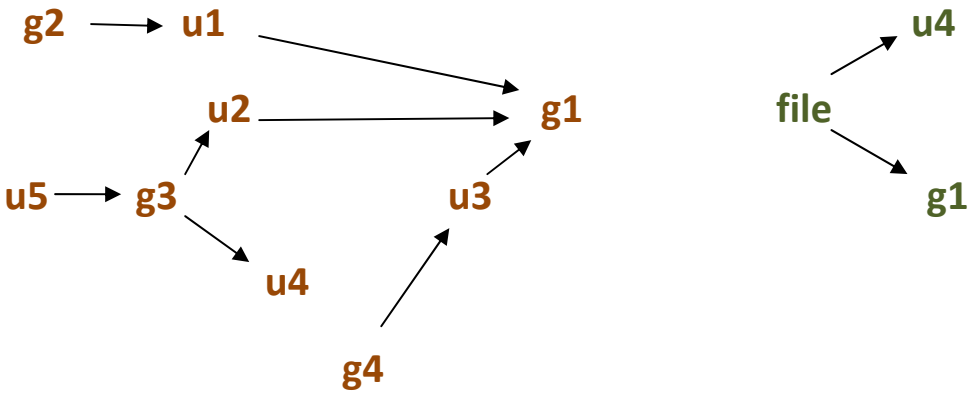
كل file له owner واحد فقط .

كل file له group واحد فقط .

ملاحظة هامة : لحظة الإنشاء يكون الـ group لملف هو group of owner

ولكن لا تختلف أبدا بينهما فيما بعد .

مثال :



- ما هي سماحيات u3 على الملف :

يجب أن نعرف السماحيات المفروضة على الملف ولتكن :

rwx	r - x	r - -
owner	group of owner	other

u3 ليس Owner

هل ينتمي لـ g1 ؟ نعم ⟵ يسمح له بـ r و x (group of owner)

- ما هي سماحيات u5 على الملف :

u5 ليس owner وليس من الـ group (لأنه ينتمي إلى g3 والـ group لهذا الملف هو g1) وبالتالي هو other : ⟵ يسمح له بـ r فقط .

< لا حظ أن u5 ينتمي إلى مجموعة الـ owner نفسها g3 ولكن هذا لا يعني شيء >

< إذا كان user بنفس الوقت owner وينتمي لـ file's group فإنه يأخذ سماحيات الـ owner والـ group >

:~>chown u2:g4 file تغيير group,owner بنفس الوقت

⇕

وهي تكافئ :

:~>chown u2 file

:~>chgrp g4 file

تغيير السماحيات : :~>chmod 0755 file

تغيير السماحيات : :~>chmod 0621 dir

لها الشكل الثاني أيضاً :

:~>chmod [ ] ± [ ] file

Owner ←	u	r
Group ←	g	w
Other ←	o	x
All ←	a	أياً منها

مـثال :

* :~>chmod u + w file

⇨ rw- r - x r - -

* :~>chmod u + wx file

⇨ rwx r - x r - -

* :~>chmod g - x file

⇨ r - - r - - r - -

ملاحظة : يمكن أن يقوم الـ owner بحذف بعض السماحيات منه مثلاً :

حذف w لجعله read only حرصاً على عدم تغييره .

:~>chmod a - rwx file

⇨ - - - - - - - - -

يمكن تغييرها لاحقاً .

==========

:~>lsess /etc/httpd/conf/httpd.conf فتح ملف

G ترد لأخر الملف

g ترد لأول الملف

/ بحث عن الكلمة في الملف من الاول الى الاخر

? بحث عن الكلمة في الملف من الأخر الى الاول

:~>vi اسم الملف انشاء ملف

i الكتابة ⟶ escap للخروج

:q ↷ الملف لم يحفظ بعد

escap : w ↷ اظهار عدد المحارف

escape :q ↷ للخروج بعد الحفظ

escap : wq ↷ للخروج مع الحفظ

escap : q! ↷ للخروج بدون الحفظ حتى بعد التعديل

بعد الدخول للملف ولنسخ السطر نضغط escap وyy

ووللصق نضغط p

dd للقص d للصق

escap 1 d ^ قص سطرين لفوق

escap 4 y ^ نسخ ٥ اسطر لفوق

:1,$s/old-word/new-word/g استبدال كل ورودات الكلمة القديمة بالجديدة

1 سطر البداية , $ سطر النهاية , g تبديل جميع الأسطر

==========

==========

Script

:~> env تعرض المتحولات على مستوى النظام

:~> myvar=7 تعريف متحول

:~> echo $myvar يطبع قيمة المتحول

:~> echo myvar يطبع المتحول

test [exp] if تعليمة شرطية مثل

:~>echo $? ⇨ { 0 if the last command is true } { 1 if false }

:~>./namefile var script تنفيذ

Example 1 :

```
$ vi demo
#!/bin/sh
#
# Script that demos, command line args
#
echo "Total number of command line argument are $#"
echo "$0 is script name"
echo "$1 is first argument"
echo "$2 is second argument"
echo "All of them are :- $* or $@"
```

:~>./demo var

Example 2 :

script يقرأ المتحولات المدخلة ويطبعها ويطبع مجموعها

vi ./count.sh

#!/bin/bash

echo" you have s# parameter"

echo" you have s* parameter"

./count.sh aaa qqq www

==========

==========

:~>wc [OPTION] [FILE] { print newline, word, and byte counts for each file}

If the option are :

-c print the byte counts

-m print the character counts

-l print the newline counts

-L print the length of the longest line

-w print the word counts

==========

Cut : {remove sections from each line of files }

:~>cut [OPTION] [FILE] DESCRIPTION

Print selected parts of lines from each FILE to standard output. Mandatory arguments to long options are mandatory for short options too

If the option are :

-f select only these fields; also print any line that contains no delimiter char? acter, unless the -s option is specified

-d --delimiter=DELIM

use DELIM instead of TAB for field delimiter

==========

==========

head output the first part of files

:~>head [OPTION] [FILE]

If the option are :

-n print the first n lines of file

==========

tail output the last part of files

:~> tail [OPTION] [FILE]

If the option are :

-n output the last n lines

==========

Edit a file and full it with a words and many line

Then try this commands :

```
:~> cat yourfile |head -5
:~> cat yourfile |tail -4
:~> cat yourfile |wc -l
:~>cat yourfile |cut -f1 -d:
:~> cat yourfile |head -4 |tail -1
:~> cat yourfile |head -5 |tail -1|cut -f1 -d:|wc -c
:~> cat yourfile |head -4 |tail -1|cut -f1 -d:|wc -l
:~> cat yourfile |head -5 |tail -1|cut -f1 -d:
```

Help File Library: Bash Programming Cheat Sheet

A quick cheat sheet for programmers who want to do shell scripting. This is not intended to teach programming, etc. but it is intended for a someone who knows one programming language to begin learning about bash scripting.

Basics

All bash scripts must tell the o/s what to use as the interpreter. The first line of any script should be:
#!/bin/bash

You must make bash scripts executable.
chmod +x filename

Variables

Create a variable - just assign value. Variables are non-datatyped (a variable can hold strings, numbers, etc. with out being defined as such).
varname=value

Access a variable by putting $ on the front of the name
echo $varname

Values passed in from the command line as arguments are accessed as $# where #= the index of the variable in the array of values being passed in. This array is base 1 not base 0.
command var1 var2 var3 .... varX
$1 contains whatever var1 was, $2 contains whatever var2 was, etc.

Built in variables:

Variable	Use
$1-$N	Stores the arguments (variables) that were passed to the shell program from the command line.
$?	Stores the exit value of the last command that was executed.
$0	Stores the first word of the entered command (the name of the shell program).
$*	Stores all the arguments that were entered on the command line ($1 $2 ...).
"$@"	Stores all the arguments that were entered on the command line, individually quoted ("$1" "$2" ...).

Quote Marks

Regular double quotes ("like these") make the shell ignore whitespace and count it all as one argument being passed or string to use. Special characters inside are still noticed/obeyed.

Single quotes 'like this' make the interpreting shell ignore all special characters in whatever string is being passed.

The back single quote marks (`command`) perform a different function. They are used when you want to use the results of a command in another command. For example, if you wanted to set the value of the variable contents equal to the list of files in the current directory, you would type the following command: contents=`ls`, the results of the ls program are put in the variable contents.

Logic and comparisons

A command called test is used to evaluate conditional expressions, such as a if-then statement that checks the entrance/exit criteria for a loop.

test expression
Or
[expression]

Numeric Comparisons

int1 -eq int2	Returns True if int1 is equal to int2.
int1 -ge int2	Returns True if int1 is greater than or equal to int2.
int1 -gt int2	Returns True if int1 is greater than int2.
int1 -le int2	Returns True if int1 is less than or equal to int2
int1 -lt int2	Returns True if int1 is less than int2
int1 -ne int2	Returns True if int1 is not equal to int2

String Comparisons

str1 = str2	Returns True if str1 is identical to str2.
str1 != str2	Returns True if str1 is not identical to str2.
str	Returns True if str is not null.
-n str	Returns True if the length of str is greater than zero.
-z str	Returns True if the length of str is equal to zero. (zero is different than null)

File Comparisons

-d filename	Returns True if file, filename is a directory.
-f filename	Returns True if file, filename is an ordinary file.
-r filename	Returns True if file, filename can be read by the process.
-s filename	Returns True if file, filename has a nonzero length.
-w filename	Returns True if file, filename can be written by the process.
-x filename	Returns True if file, filename is executable.

Expression Comparisons

!expression	Returns true if expression is not true
expr1 -a expr2	Returns True if expr1 and expr2 are true. (&& , and)
expr1 -o expr2	Returns True if expr1 or expr2 is true. (||, or)

Logic Con't.

If...then

```
if [ expression ]
        then
                commands
fi
```

If..then...else

```
if [ expression ]
        then
                commands
        else
                commands
fi
```

If..then...else If...else

```
if [ expression ]
        then
                commands
elif [ expression2 ]
        then
                commands
else
                commands
fi
```

Case select

```
case string1 in
        str1)
                commands;;
        str2)
                commands;;
        *)
                commands;;
esac
```

string1 is compared to str1 and str2. If one of these strings matches string1, the commands up until the double semicolon (; ;) are executed. If neither str1 nor str2 matches string1, the commands associated with the asterisk are executed. This is the default case condition because the asterisk matches all strings.

Iteration (Loops)

```
for var1 in list
do
        commands
done
```

This executes once for each item in the list. This list can be a variable that contains several words separated by spaces (such as output from ls or cat), or it can be a list of values that is typed directly into the statement. Each time through the loop, the variable var1 is assigned the current item in the list, until the last one is reached.

```
while [ expression ]
do
        commands
done

until [ expression ]
do
        commands
done
```

Functions

Create a function:

```
fname(){
        commands
}
```

Call it by using the following syntax: fname

Or, create a function that accepts arguments:

```
fname2 (arg1,arg2...argN){
        commands
}
```

And call it with: fname2 arg1 arg2 ... argN