Introduction to Unix/Linux

Part 2

Anita Schwartz
Client Support & Services

Variables and Environments

Variables are used to help control your environment. Each shell keeps track of its own shell and environment variables to maintain your environment.

- **Environment or global variables**

  Variables defined for the current shell and are inherited by any child shell. Basically available in all shells.

- **Shell or local variables**

  Variables only available in the current shell.

Variables and Environments

Typically shell and environment variables are defined with all capital lettters. You cannot use a number as the first character of any variable.

- Use command **`printenv`** or **`env`** to list current values of all environment variables.

- Use command **`set`** to list all shell variables, environment variables, local variables and shell functions.

Variables and Environments

Variables are defined using

```
    VAR_NAME=value:value
```

Or

```
    VAR_NAME="string with spaces"
```

No spaces around the `=`

Variables and Environments: Exercise

Try each of the following commands

`printenv` or **`env`** to see your environment variables.

`set | less` to see all shell variables, environmental variables, local variables and shell functions.

The **`|`** (pipe) is used to redirect the output from the command **`set`** to the program **`less`** to display one page at a time. This is helpful when you have a lot of output displayed from a command.

Common Environment and Shell Variables

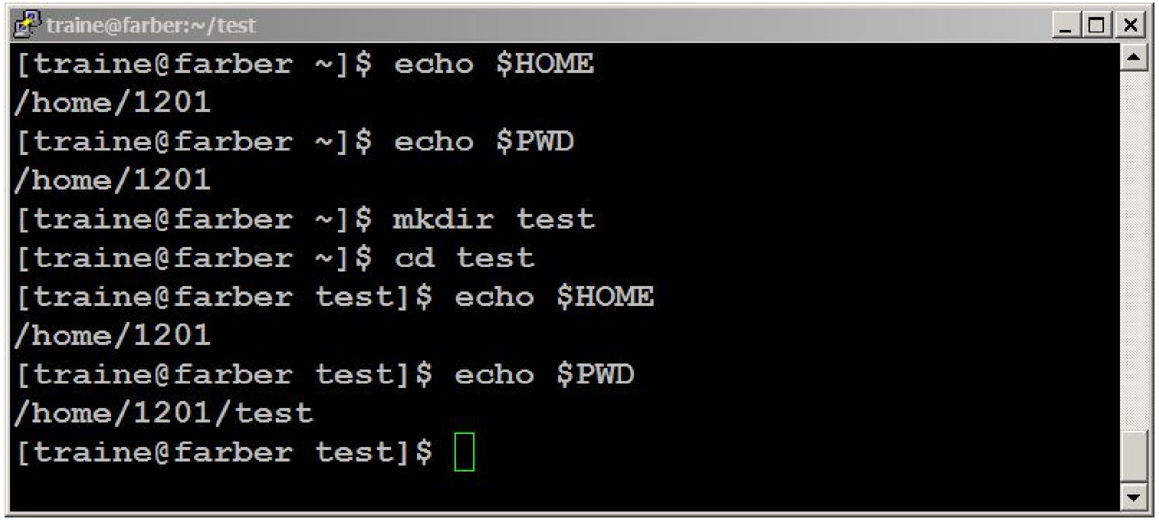
Use command **`echo $VAR_NAME`** to display the current value of the variable, where $VAR_NAME might be

- `SHELL`
- `HOME`
- `PWD`
- `BASH`

Variables and Environments: Exercise

Try

```
echo $HOME

echo $PWD

mkdir test

cd test

echo $HOME

echo $PWD
```



```
[traine@farber ~]$ echo $HOME
/home/1201
[traine@farber ~]$ echo $PWD
/home/1201
[traine@farber ~]$ mkdir test
[traine@farber ~]$ cd test
[traine@farber test]$ echo $HOME
/home/1201
[traine@farber test]$ echo $PWD
/home/1201/test
[traine@farber test]$
```

Variables and Environments

Every time a new shell is started, environment variables (list from **`printenv`** or **`env`** command) become available in the new shell (sub-shell), basically a copy of the environment. Shell or local variables are only available in the current shell, and not available in the sub-shell.

Login shell

```
$ MY_VAR="A local var"
$ echo $MY_VAR
A local var
$ bash
$ echo $MY_VAR
```

First sub-shell

No MY_VAR; create it

```
$ MY_VAR="A local var in
sub-shell"
$ echo $MY_VAR
A local var in sub-shell
$ export MY_VAR
```

Make it global for future sub-shell

Second sub-shell

```
$ bash
$ echo $MY_VAR
A local var in sub-shell
$ unset MY_VAR
$ echo $MY_VAR
```

Remove MY_VAR

```
$ exit
$ echo $MY_VAR
A local var in sub-shell
$ exit
$ echo $MY_VAR
A local var
```

Back to First sub-shell

Back to login shell

Exercise: Creating a Shell Variable

Try

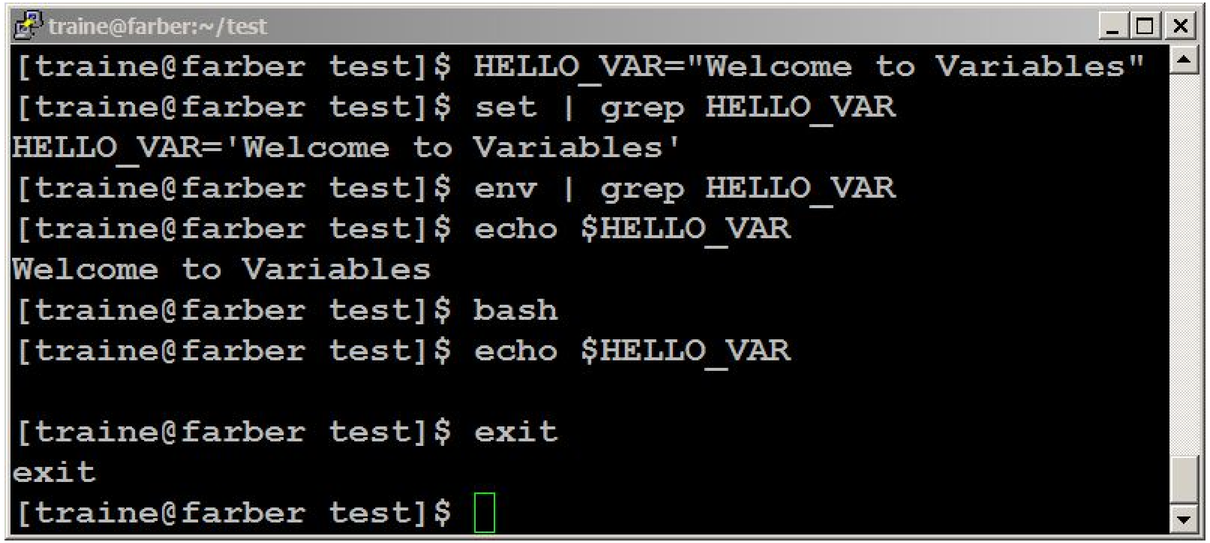
```
HELLO_VAR="Welcome to Variables"

set | grep HELLO_VAR

env | grep HELLO_VAR

echo $HELLO_VAR
```

The `|` (pipe) is used to redirect the output from the command **`set`** and **`env`** to the program **`grep`** to search for the pattern **`HELLO_VAR`** and only display the lines that contain it. This is helpful to customize your output to only display what you need.

Exercise: Creating a Shell Variable

Try

```
bash

echo $HELLO_VAR

exit
```



```
[traine@farber test]$ HELLO_VAR="Welcome to Variables"
[traine@farber test]$ set | grep HELLO_VAR
HELLO_VAR='Welcome to Variables'
[traine@farber test]$ env | grep HELLO_VAR
[traine@farber test]$ echo $HELLO_VAR
Welcome to Variables
[traine@farber test]$ bash
[traine@farber test]$ echo $HELLO_VAR

[traine@farber test]$ exit
exit
[traine@farber test]$ 
```

Exercise: Creating an Environment Variable

Try

```
export HELLO_VAR

env | grep HELLO_VAR

bash

echo $HELLO_VAR

exit
```

Removing Variables

Use command **`unset VAR_NAME`**

Aliases

Typically used to help customize commands you want to use with common options or default values.

Use command **`alias`** to see the list of aliases.

Aliases: Exercise

Try

```
alias
```

You might see something like this:

```
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
```

Try

```
ll
```

Exercise: Create your own alias

Try

```
alias myscratch=”cd /lustre/scratch/anita”

alias | grep myscratch

pwd

myscratch

pwd
```

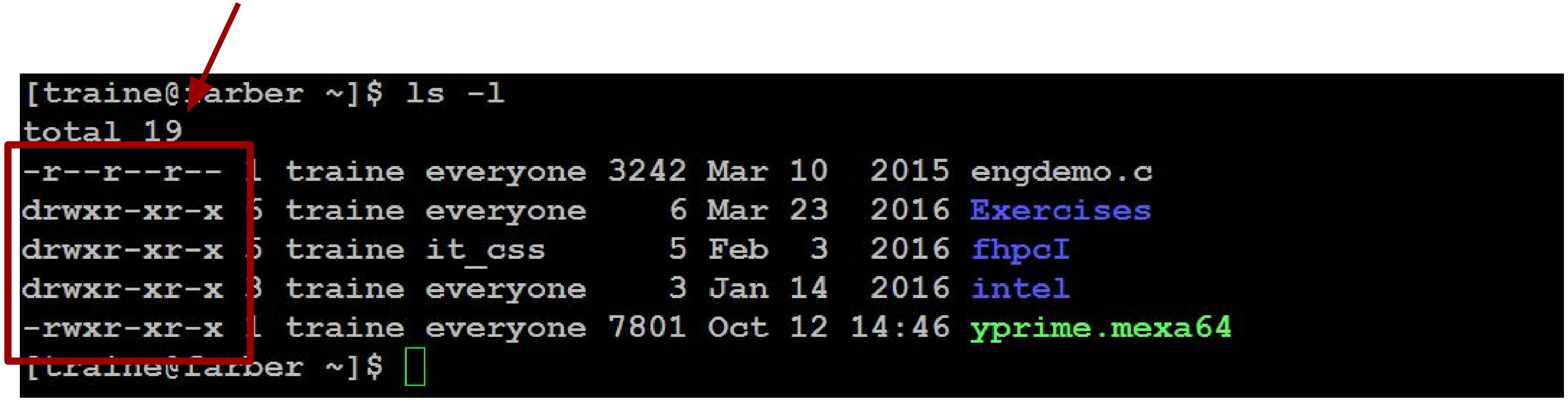
Permissions

Every file in Unix/Linux have permissions based on the following attributes

- **type**: indicates file type
- **user**: owner of the file (the user who created the file)
- **group**: any users who belong is the same group as the user who created the file will have these permissions
- **other**: any user who is outside the group will have these permissions to the file

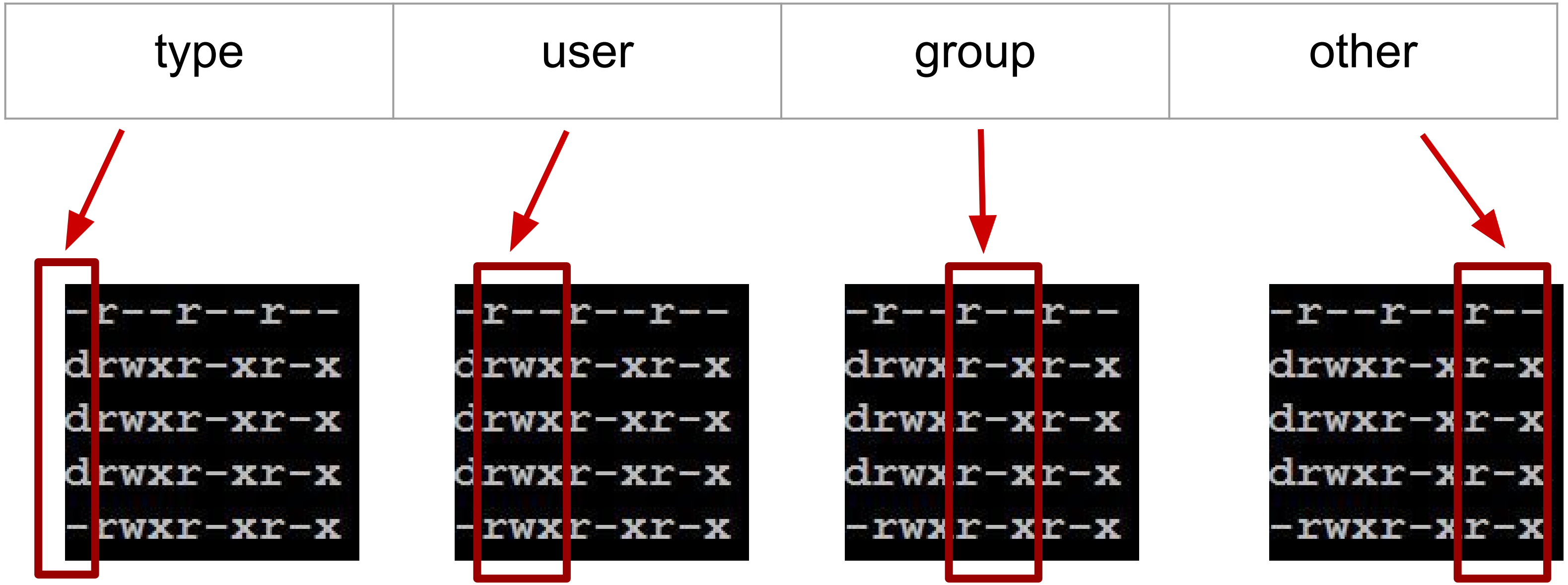
Permissions

Using the command `ls -l` displays a list of files and their permissions.



```
[traine@farber ~]$ ls -l
total 19
-r--r--r-- 1 traine everyone 3242 Mar 10  2015 engdemo.c
drwxr-xr-x 5 traine everyone    6 Mar 23  2016 Exercises
drwxr-xr-x 5 traine it_css      5 Feb  3  2016 fhpcI
drwxr-xr-x 3 traine everyone    3 Jan 14  2016 intel
-rwxr-xr-x 1 traine everyone 7801 Oct 12 14:46 yprime.mexa64
[traine@farber ~]$
```

Permissions



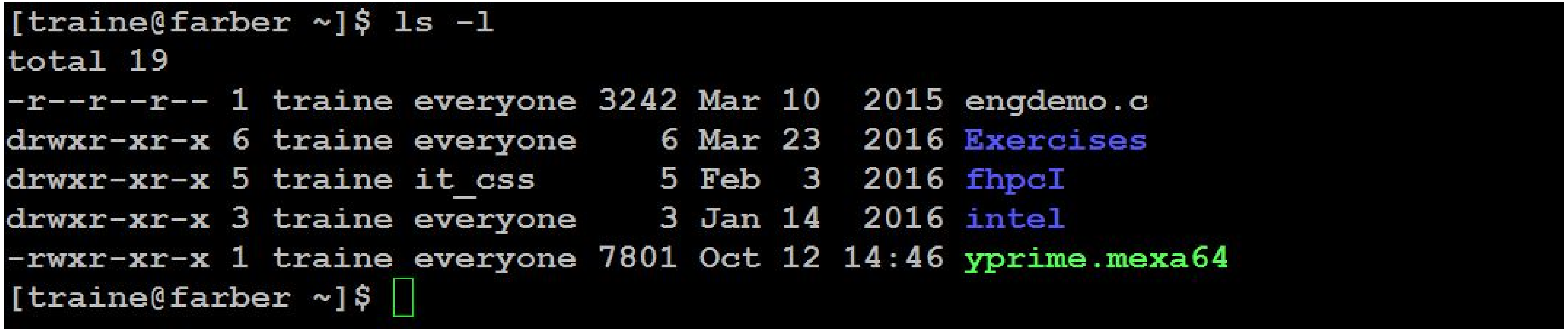
```
-r--r--r--
drwxr-xr-x
drwxr-xr-x
drwxr-xr-x
-rwxr-xr-x
```

`-`=regular file, `d`=directory, `l`=link

`r`=read, `w`=write, `x`=executable (or searchable if type=`d`)

Exercise: Permissions

Let’s examine in detail some of the files in the listing:



```
[traine@farber ~]$ ls -l
total 19
-r--r--r-- 1 traine everyone 3242 Mar 10  2015 engdemo.c
drwxr-xr-x 6 traine everyone    6 Mar 23  2016 Exercises
drwxr-xr-x 5 traine it_css      5 Feb  3  2016 fhpcI
drwxr-xr-x 3 traine everyone    3 Jan 14  2016 intel
-rwxr-xr-x 1 traine everyone 7801 Oct 12 14:46 yprime.mexa64
[traine@farber ~]$ 
```

`engdemo.c`: is a regular file (`-`) and user, group and other all have read (`r`) access only.
`Exercises`: is a directory (`d`) and user has read, write, searchable (`rwx`) access, group and other have read and searchable (`r-x`) access.
`yprime.mexa64`: is a regular file and user has read, write, executable (`rwx`) access, group and other have read and executable (`r-x`) access.

Change Permissions

Use `chmod` command to change permissions using two different methods:

- Letters: `a` (all (everyone)), `u` (user), `g` (group) and `o` (other)

  use a `+` or `-` (plus or minus sign) to add or remove permissions for a file respectively. Use an equals sign `=`, to specify new permissions and remove the old ones for the particular type of user(s).

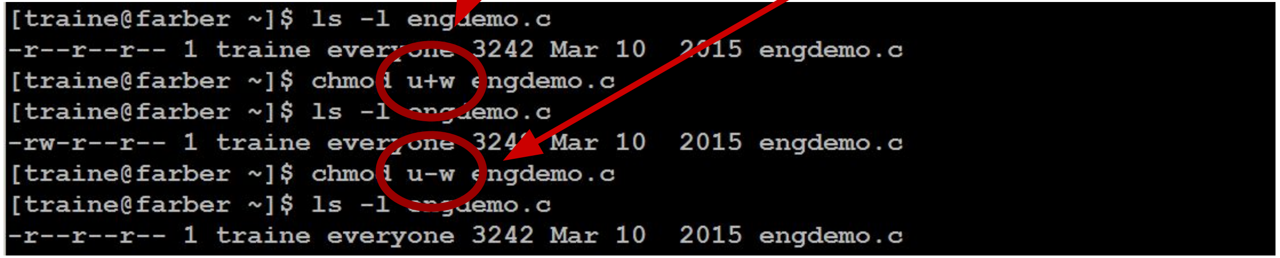
- Numbers: `r` (read) = 4, `w` (write) = 2, `x` (execute) = 1

Use **`man chmod`** to get help on the **`chmod`** command.

Exercise: Change Permissions

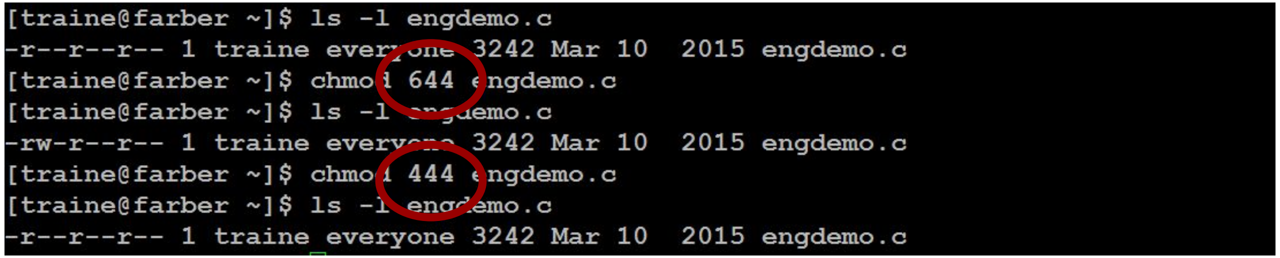
What if we wanted to change the permissions on the file `engdemo.c` so the user has write (`rw-`) permissions too?

Letter method:



```
[traine@farber ~]$ ls -l engdemo.c
-r--r--r-- 1 traine everyone 3242 Mar 10  2015 engdemo.c
[traine@farber ~]$ chmod u+w engdemo.c
[traine@farber ~]$ ls -l engdemo.c
-rw-r--r-- 1 traine everyone 3242 Mar 10  2015 engdemo.c
[traine@farber ~]$ chmod u-w engdemo.c
[traine@farber ~]$ ls -l engdemo.c
-r--r--r-- 1 traine everyone 3242 Mar 10  2015 engdemo.c
```

Number method:



```
[traine@farber ~]$ ls -l engdemo.c
-r--r--r-- 1 traine everyone 3242 Mar 10  2015 engdemo.c
[traine@farber ~]$ chmod 644 engdemo.c
[traine@farber ~]$ ls -l engdemo.c
-rw-r--r-- 1 traine everyone 3242 Mar 10  2015 engdemo.c
[traine@farber ~]$ chmod 444 engdemo.c
[traine@farber ~]$ ls -l engdemo.c
-r--r--r-- 1 traine everyone 3242 Mar 10  2015 engdemo.c
```

Permissions

More details available at

https://www.tutorialspoint.com/unix/unix-file-permission.htm

University of Delaware

Information Technology
Research Computing

Questions and Open Forum