Introduction to bash

(recap)

What is bash?

When you are using a terminal in cygwin, Mac OS X, or GNU/Linux (or other Unix-like environments), you are presented with a prompt and a blinking cursor.

The terminal is waiting for you to enter some commands (and hit ENTER).

The program which is handling your input is called bash.

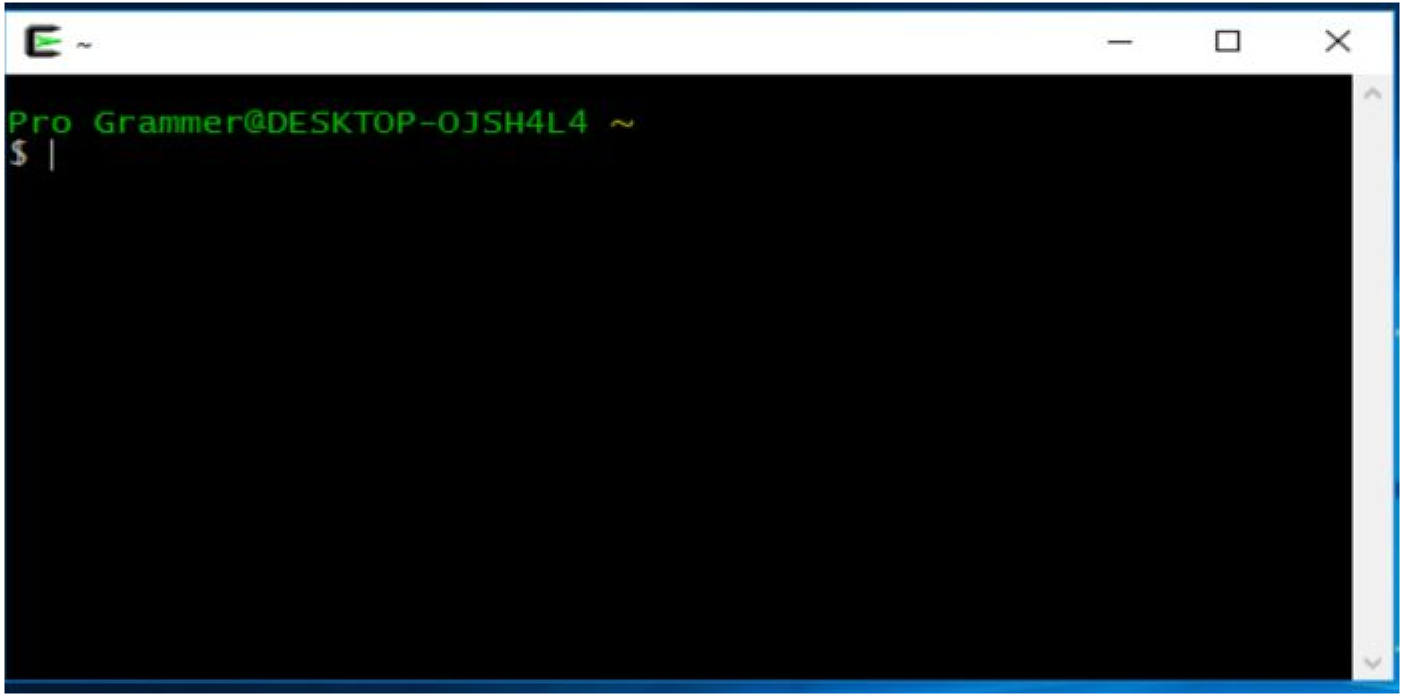
You can call bash a “shell” or a “command interpreter”. It interprets your commands!

A terminal running bash

prompt:

user@comp “path”

$ (cursor)



```
Pro Grammer@DESKTOP-OJSH4L4 ~
$ |
```

A few first commands

Listing files: `ls` (mnemonic: LiSt - LS)

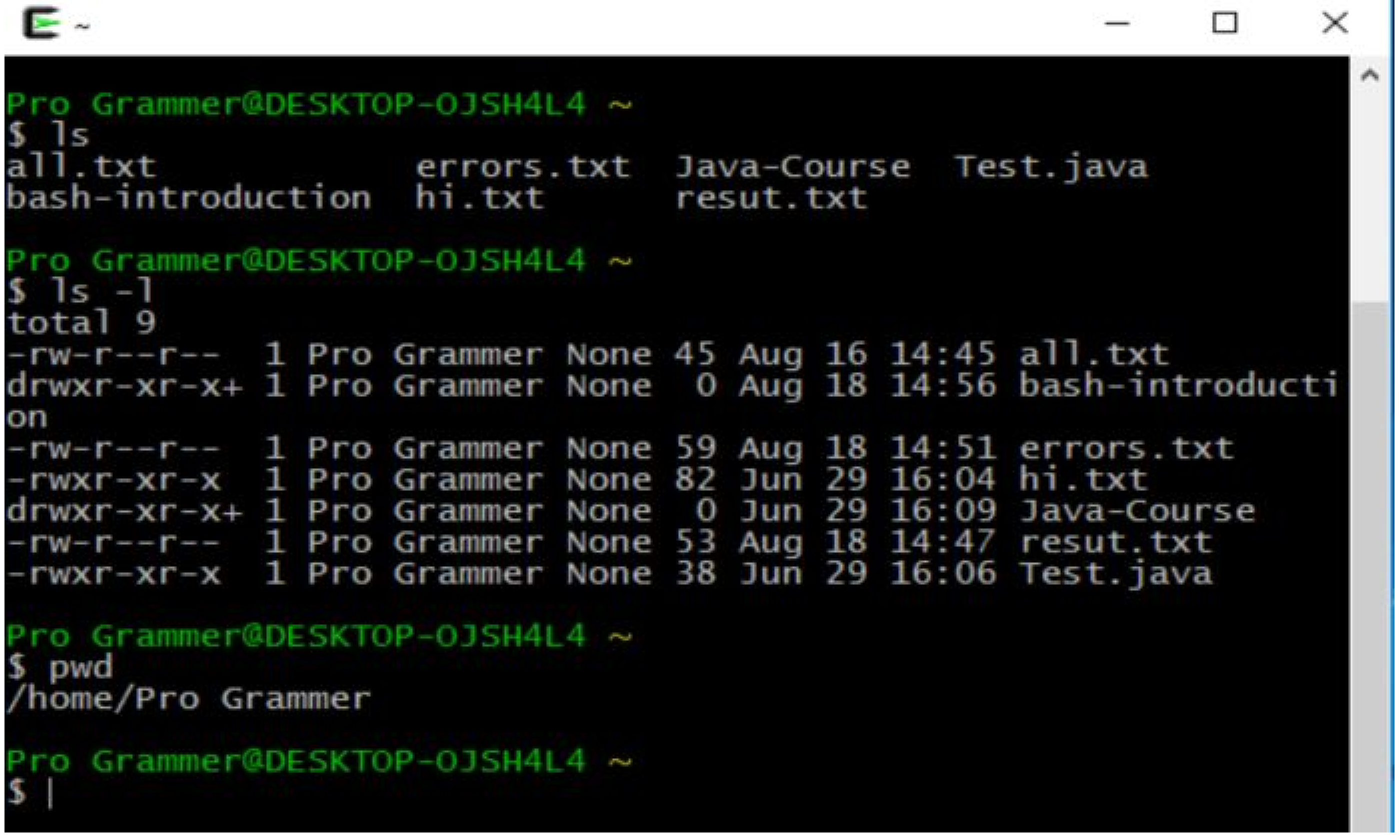
Listing files with long descritption: `ls -l` (the “`-l`” part is an “option” or “flag”)

Listing all files: `ls -a` (including files starting with a dot, which are hidden by default from `ls`)

Combining options: `ls -al` (all files, long listing, same as `ls -a -l`)

What is the current directory? `pwd` (print working directory)

Running ls and pwd



```
~

Pro Grammer@DESKTOP-OJSH4L4 ~
$ ls
all.txt             errors.txt  Java-Course  Test.java
bash-introduction   hi.txt      resut.txt

Pro Grammer@DESKTOP-OJSH4L4 ~
$ ls -l
total 9
-rw-r--r--  1 Pro Grammer None 45 Aug 16 14:45 all.txt
drwxr-xr-x+ 1 Pro Grammer None  0 Aug 18 14:56 bash-introducti
on
-rw-r--r--  1 Pro Grammer None 59 Aug 18 14:51 errors.txt
-rwxr-xr-x  1 Pro Grammer None 82 Jun 29 16:04 hi.txt
drwxr-xr-x+ 1 Pro Grammer None  0 Jun 29 16:09 Java-Course
-rw-r--r--  1 Pro Grammer None 53 Aug 18 14:47 resut.txt
-rwxr-xr-x  1 Pro Grammer None 38 Jun 29 16:06 Test.java

Pro Grammer@DESKTOP-OJSH4L4 ~
$ pwd
/home/Pro Grammer

Pro Grammer@DESKTOP-OJSH4L4 ~
$ |
```

Filesystem and bash

There is always a “working directory” (a.k.a. “current directory”)

Running `pwd` tells us our current directory. Running `ls` lists the files of the current directory.

When we open a terminal, bash drops us in our “home directory”.

The top level directory is called /

Every directory can be described from /

The separator between directories in such a path is also “/”

Home directory

Every user on your system has a home directory. The path to the home directory is usually `/home/username/` where “username” is your user name:)

Bash uses a nickname for the home directory, ~ (pronounced “tilde”).

The prompt in cygwin when you start a new terminal shows that you are in the home directory:

```
username@computername ~    ← Note the tilde!
$ pwd
/home/username
```

Prompt on a mac

To be kind to the mac users, here’s how the prompt usually look on a mac:

```
ComputerName:~ UserName$ _
```

The $

Note that the $ only indicates that you are running bash. When we say in an exercise “do this:”

```
$ ls
```

Then we don’t mean that you should also type in the $ (it is not part of the command, but there to indicate that it is a bash command)

Changing directory

We always have a working directory, and sometimes we say “You are in a directory”. When you start the terminal, “you are in the home directory”.

It is possible to change the working directory, or “to change directory”. You use the command `cd` to do this. Mnemonic: “ChangeDirectory”.

To move from the home directory to the root directory / you type this:

```
$ cd /
```

To change directory back to your home, you can do this:

```
$ cd ~
```

Go home!

There are a few ways to go back to the home directory. First using the absolute path (the path all the way from the root directory):

```
$ cd /home/username
```

Second, you can use the nickname ~

```
$ cd ~
```

Third, actually, if you don’t give any argument to cd, you will go home!

```
$ cd
```

Go back! (to where you once belonged)

A convenient way to alter between two directories, is to give cd the argument - (a dash, or as some say “a minus”), which means “go to the previous directory”

```
$ pwd
/home/username
$ cd /
$ pwd
/
$ cd -
$ pwd
/home/username
$ cd -
$ pwd
/
```

Creating directories

You can use the mkdir command to create a directory:

```
$ mkdir textfiles
$ cd textfiles
```

Note that we can change directory to textfiles without an absolute path starting from /

This is called a relative path (meaning a path from the current directory)

Relative path

Going to a directory created in current directory is simple, the path to the directory is the directory name itself:

```
$ cd textfiles
```

This is called “going down” to textfiles. To go back (go “up” again) we can use the nickname for the directory immediately above current directory which is .. (dot-dot):

```
$ cd ..
```

These are relative paths.

Relative paths - up and down

Current directory also has a nick name, “.” (a single dot).

If we have the following directories under the current directory:

```
.
|-- letters
|-- notes
`-- poems
```

That means that there are three directories in the current directory. We can go down to e.g. letters using `$ cd letters` and back up again using

```
$ cd ..
```

Relative to what?

A relative path is always a path relative to the current directory. Going down in directories under the current directory is straightforward, simply type the path starting with the first directory and use / to separate directories.

The path from / to your home directory is:

```
home/username
```

Going up using relative paths, means using .. for each directory “above”. The relative path to / from your home is:

../..   (meaning up one directory to /home then one more to /)

Absolute paths

Absolute paths always start with /

Since in bash, there is only one root directory (/), every directory and file can be described by a path from the root and down to the file or directory.

Examples:

/home/username/textfiles/notes/shoppinglist.txt

/home/username/java/exercises/first-program/HelloWorld.java

It doesn’t matter what the current directory is, an absolute path always works (if it is correct).

Deleting directories

An empty directory can be deleting using the command `rmdir`:

```
$ rmdir textfiles/poems   #(there were no poems written :/ )
```

If the directory turns out to be non-empty, you will get an error message warning you that the directory wasn’t deleted because it had files in it.

If you want to delete a directory and all its contents (including subdirectories and all their files) you may use the rm command instead with a flag:

```
$ rm -r textfiles/poems #(poems and everything “under” it!)
```

Use with care. You will get asked for each file etc, but there is another flag...

Recap of the nicknames

~	home directory of the logged in user
.	current directory (you can think “here”)
..	one directory above

Deleting files

To delete a single file, you can use the `rm` command:

```
$ rm textfiles/poems/jabberwocky.txt
```

Arguments and options/flags

A command may accept arguments. Some commands make no sense without arguments. Arguments are information you provide so that the command can complete its task. Example:

```
$ mkdir
mkdir: missing operand
Try 'mkdir --help' for more information.
```

Make directory… you need to help the command here, make what directory?

Arguments and options/flags

Some commands have optional instructions which changes something about the task. To list files in a long format, you can use this:

```
$ ls -l
```

The flag is "`–l`" and it tells ls *how* to list the files. To list the files in textfiles in a long listing, you provide both an argument (what to list) and an option (how to list it):

```
$ ls -l textfiles
```

Some exercises

Using echo with flags:

```
$ echo "Hello bash"
Hello bash
$ echo -n "Hello bash"
Hello bash$ echo "Hello\nbash"
Hello\bash
$ echo -e "Hello\nbash"          # -e cares about escapes
Hello                            # \n means newline
bash
$ echo -e "Hello\rGood bye"      # -r carriage return
Good bye
```

What does this do?

Combining the flags -e (use escapes) and -n (don’t do a newline):

```
for i in {1..10} # repeat 10 times the following
do
 echo -ne "\r|"
 sleep 0.1
 echo -ne '\r/'
 sleep 0.1
 echo -ne "\r-"
 sleep 0.1
 echo -ne "\r\\"
 sleep 0.1
done
```