

BASH Shell Command Grouping

Commands can be executed on the command line in different ways

Single Commands

Syntax:

\$bash command [-options] [arguments]

\$bash ls -lap /etc

Notes

The command runs in the foreground

Prompt returns when command is complete

An exit code of 0 is returned if command is successful

An exit code other than 0 is returned if command is unsuccessful

Multiple Commands (separated by ;)

Syntax:

\$bash command1 ; command2 ; ... command_n

\$bash cal ; ls -la ; pwd ; who

Notes

Each command runs in the foreground

Prompt returns when last command is complete

Each command returns an exit code of 0 if command is successful

Each command returns an exit code other than 0 if command is unsuccessful

Each command is treated as a separate process and has a different PID

Each subsequent command will attempt to execute whether the previous command was successful or not

Multiple Commands with Pipes

Syntax:

```
$bash command1 | command2 | ... | command_n
```

```
$bash cat myData | sort | less
```

Notes

Output of command1 is input to command2

Commands run sequentially by the shell

Prompt returns when last command is complete

Commands execute as part of one process

An exit code is generated after the last command is run

The command line has one PID

The & Operator

Syntax:

```
$bash command1 & command2 &
```

```
$bash cat myData & sort &
```

Notes

Commands followed by the & operator run in the background

Prompt returns immediately

Commands execute as independent processes

Each command line has a different PID
kill PID_num is required to terminate a background process

The && Operator

Syntax:

```
$bash command1 && command2
```

```
$bash pwd ; ls -la && cat myData | sort
```

Commands to the RIGHT of the && operator are executed ONLY if all commands to the LEFT of the && operator terminate successfully

If there is a typo:

```
$bash pwdx ; ls -la && cat myData | sort
```

pwdx will fail, so *cat myData | sort* will not be executed

The || Operator

Syntax:

```
$bash command1 || command2
```

```
$bash pwd ; ls -la || cat myData | sort
```

Commands to the RIGHT of the || operator are executed ONLY if at least ONE of the commands to the LEFT of the || operator terminate *unsuccessfully*

Given:

```
$bash pwd ; ls -la || cat myData | sort
```

pwd ; ls -la should work, so cat myData / sort will not be executed

However,

Given:

```
$bash pwdx ; ls -la || cat myData | sort
```

pwdx ; ls -la will fail, so cat myData / sort will be executed

The (list;) Group Command

Syntax:

```
$bash ( command1 ; command2 ; command3 )
```

```
$bash ( pwd ; ls -la /etc ; cat myFil )
```

Notes

The group of commands within the parens run in a subshell

Commands execute as one collective process so the group has one PID

The entire group can collectively be run in the background and have only one PID

Example:

```
$bash ( pwd ; ls -la /etc ; cat myFil ) &
```

The { list; } Group Command

Syntax:

```
$bash { command1 ; command2 ; command3; }
```

```
$bash { pwd ; ls -la /etc ; cat myFil; }
```

Notes

The group of commands within the curly braces run in foreground of the current shell

This means that the output of the group of commands within the curly braces can be fed to another operator outside the curly braces

Example:

```
$bash { ls -la /etc ; cat myFil herFil; } | less
```

Notes on Variable Assignments and Command Grouping

Given:

```
$ myVar=5    # the value 5 is assigned to myVar
```

```
$ echo $myVar  
5
```

```
$ (myVar=24) # spawn a new shell and assign 24 to  
# myVar
```

```
$ echo $myVar  
5                # huh?????? what happened?
```

```
$ { myVar=57; } # the value 57 is assigned to myVar  
# in the current shell
```

```
$ echo $myVar  
57
```