Variable settings

Basic Variable Setting (=) 2
How to check the value of variable? 2
Line Joining 2
Setting a default value(?=) 3
    What's the benefit? 3
    What happens if we have multiple ?= 3
Setting a weaker default value (??=) 3
Difference between ?= and ??= 3
Variable Expansion 4
What happens if C is not defined in above? 4
Immediate Variable Expansion (:=) 4
Appending operators 5
    += 5
    .= 5
Difference between += and .= is space is automatically added in += 5
Prepending Operators 5
    =+ 5
    =. 5
Appending and Prepending (Override Style Syntax) 5
Removal 5
Override Style Operation Advantages 6

Basic Variable Setting (=)

VARIABLE = "value"

The following example sets VARIABLE to "value".

Eg. MACHINE = "raspberrypi3"

This assignment occurs immediately as the statement is parsed

It is a "hard" assignment

If you include leading or trailing spaces as part of an assignment, the spaces are retained:

VARIABLE = " value"

VARIABLE = "value "

Note: You can also use single quotes ('') instead of double quotes when setting a variable's value

Benefit:

VARIABLE = 'I have a " in my value'

How to check the value of variable?

For configuration changes, use the following:

```
$ bitbake -e
```

This command displays variable values after the configuration files (i.e. local.conf, bblayers.conf, bitbake.conf and so forth) have been parsed.

For recipe changes, use the following

```
$ bitbake recipe -e | grep VARIABLE="
```

Line Joining

BitBake joins any line ending in a backslash character ("\") with the following line before parsing statements

The most common use for the "\" character is to split variable assignments over multiple lines

```
BBLAYERS ?= " \
  /home/linuxtrainer/Yocto_Training/source/poky/meta \
  /home/linuxtrainer/Yocto_Training/source/poky/meta-poky \
  /home/linuxtrainer/Yocto_Training/source/poky/meta-yocto-bsp \
  "
```

```
$ bitbake -e | grep ^BBLAYERS
```

Setting a default value(?=)

?= is used for soft assignment for a variable

What's the benefit?

Allows you to define a variable if it is undefined when the statement is parsed,

If the variable has a value, then the soft assignment is lost

Eg: MACHINE ?= "qemuarm"

If MACHINE is already set before this statement is parsed, the above value is not assigned

If MACHINE is not set, then the above value is assigned

Note: Assignment is immediate

What happens if we have multiple ?=

If multiple "?=" assignments to a single variable exist, the first of those ends up getting used

Setting a weaker default value (??=)

Weaker default value is achieved using the ??= operator

Difference between ?= and ??=

Assignment is made at the end of the parsing process rather than immediately.

When multiple "??=" assignments exist, the last one is used

Eg.

MACHINE ??= "qemux86"

MACHINE ??= "qemuarm"

If MACHINE is not set, the value of MACHINE = "qemuarm"

If MACHINE is set, before the statements, then the value will not be changed

It is called weak assignment, as assignment does not occur until the end of the parsing process.

Note: "=" or "?=" assignment will override the value set with "??="

Variable Expansion

Variables can reference the contents of other variables using a syntax that is similar to variable expansion in Bourne shells

```
A = "hello"

B = "${A} world"


$ bitbake -e | grep ^A=

$ bitbake -e | grep ^B=
```

The "=" operator does not immediately expand variable references in the right-hand side

Instead, expansion is deferred until the variable assigned to is actually used

```
A = "${B} hello"

B = "${C} world"

C = "linux"

$ bitbake -e | grep ^A=
```

What happens if C is not defined in above?

the string is kept as is

Immediate Variable Expansion (:=)

The ":=" operator results in a variable's contents being expanded immediately, rather than when the variable is actually used

```
A = "11"

B = "B:${A}"

A = "22"

C := "C:${A}"

D = "${B}"

A = "11"

B := "B:${A}"

A = "22"

C := "C:${A}"

D = "${B}"
```

Appending operators

+=

A = "hello"   A += "world"

.=

A = "hello" A .= "world"

Difference between += and .= is space is automatically added in +=

These operators take immediate effect during parsing

Prepending Operators

=+

A = "world"

A =+ "hello"

=.

A = "world"

A =. "hello"

Same as previous, =+ adds an additional space

Appending and Prepending (Override Style Syntax)

You can also append and prepend a variable's value using an override style syntax.

When you use this syntax, no spaces are inserted.

A = "hello"

A_append = " world"

B = "test"

B_append = "world"

C = "full"

C_prepend = "house"

Removal

You can remove values from lists using the removal override style syntax

Specifying a value for removal causes all occurrences of that value to be removed from the variable.

FOO = "123 456 789 123456 123 456 123 456"

```
FOO_remove = "123"
```

```
$ bitbake -e | grep ^FOO=
```

Override Style Operation Advantages

An advantage of the override style operations "_append", "_prepend", and "_remove" as compared to the "+=" and "=+" operators is that the override style operators provide guaranteed operations

```
IMAGE_INSTALL += "usbutils"          IMAGE_INSTALL_append = " usbutils"
```