Other great stuff

xargs

build and execute command lines from standard input

-n use at most max_args arguments per line

```
eureka@ubuntu:~$ echo 1 2 3 4
1 2 3 4
```

```
eureka@ubuntu:~$ echo 1 2 3 4 |
xargs -n2
1 2
3 4
```

-P – run up to max_proc at a time

```
find /directory  -mindepth 2 -name "*file.csv" |xargs -n 2 -P 4 grep
“hi”
```

Process substitution

A form of inter-process communication that allows the input or output of a command to appear as a file

```
eureka@ubuntu:~$ diff <(sort test_file) <(sort other_test_file)
1,4c1,4
< 1 CHI*,b
< 2 LON,c
< 3 TOK,a
< 4 SAO,a
---
> 1 yes
> 2 no
> 3 yes
> 4 no
```

Shell expansion

Expansion is performed on the command line after it has been split into tokens. There are seven kinds of expansion performed:

More info: http://linuxcommand.org/lc3_lts0080.php

- Brace Expansion: Expansion of expressions within braces.
- Tilde Expansion: Expansion of the ~ character.
- Shell Parameter Expansion: How Bash expands variables to their values.

• Command Substitution: Using the output of a command as an argument.

• Arithmetic Expansion: How to use arithmetic in shell expansions.

• Process Substitution: A way to write and read to and from a command.

• Word Splitting: How the results of expansion are split into separate arguments. *(default* <space><tab><newline>)

• Filename Expansion: A shorthand for specifying filenames matching patterns. Ex: *(using * or ?)*

• Quote Removal: How and when quote characters are removed from words.

Conditional operations

if [condition1 is true **]**
then
execute these commands
elif [condition2 is true **]**
then
execute these commands
else
execute those commands
fi

Flow Control

while commands; do commands; done

for variable [in words]; do

commands

done

The for loop uses IFS to to determine a new element and this often includes spaces which may return unexpected results. By default IFS is white space. You can set IFS to newline, just be sure to unset or return IFS to its original value. Or use a read command to get around this issue since it reads one line of data.

```
oldfile=$IFS; IFS=$'\n';for i in $(grep -i ""name"" /text.txt); do
echo $i; done; IFS=$oldfile
```

```
grep -i ""name"" /text.txt|while read I; do echo $I; done"
```

You can print IFS(Interal Field Seperator). Notice the use of here-strings (<<<). A here string can be considered a stripped-down form of a here document.
It consists of nothing more than COMMAND <<< $WORD,
where $WORD is expanded and fed to the stdin of COMMAND

The **nice thing about <<< is that it can be recalled in history.**

```
eureka@ubuntu:~$ cat -etv <<<"$IFS"
$
```

Control Operators

&& (AND) and || (OR) operators work like the [[ ]] compound command.

command1 && command2

command1 || command2

Compound Command

acts as an enhanced replacement for test. Evaluates to true or false.

[[expression]]

```
eureka@ubuntu:~$ x=foo
eureka@ubuntu:~$ if [[ $x == foo ]]; then
> echo "x is foo"
> fi
x is foo
```