There are cases in bash where you can't use pipes:

some\_command | some\_other\_command

because pipes introduce subshells for each component of the pipeline, when the subshells exit, any side-effects you were relying on would disappear. For example, this contrived example:

cat file | while read line; do ((count++)); done

echo $count

will display a blank line, because the $count variable does not exist in the current shell.

A bash [process substitution](https://www.gnu.org/software/bash/manual/bashref.html" \l "Process-Substitution) allows you to avoid this conundrum by allowing you to read from the "some\_command" output like you would from a file

while read line; do ((count++)); done < <(cat file)

# ....................................1.2

echo $count # the variable \*does\* exist in the current shell

(1) is a normal input redirection. (2) is the start of the <() process substitution syntax.