Minishell Test Cases

1. Basic Commands

Simple Commands

bash

```
ls
pwd
echo hello
cat /etc/hosts
```

Commands with Arguments

bash

```
ls -la
grep "test" file.txt
echo -n "no newline"
wc -l file.txt
```

Command Not Found

bash

```
nonexistentcommand
asdfjkl
```

2. Built-in Commands

echo

bash

```
echo hello world
echo "hello world"
echo 'hello world'
echo -n hello
echo -n -n -n hello
echo -nnnn hello
echo $PATH
echo "$USER is logged in"
```

cd

```bash
cd /tmp
cd ..
cd
cd ~
cd /nonexistent
cd -
cd ""
cd /root  # should fail without permissions
```

pwd

```bash
pwd
cd /tmp && pwd
```

export

```bash
export VAR=value
export VAR="value with spaces"
export VAR=value1 VAR2=value2
export VAR
export
export "INVALID VAR=value"
export 123VAR=value
export =value
export VAR=
```

unset

```bash
unset VAR
unset VAR1 VAR2 VAR3
unset PATH
unset
unset NONEXISTENT
```

env

```bash
env
export TEST=123 && env | grep TEST
```

exit

```bash
exit
exit 0
exit 42
exit 255
exit 256
exit -1
exit hello
exit 1 2 3  # too many arguments
```

3. Quotes

Single Quotes

```bash
echo 'hello world'
echo '$USER'
echo 'test "double" quotes'
```

Double Quotes

```bash
echo "hello world"
echo "$USER"
echo "test 'single' quotes"
echo "$PATH is the path"
```

Mixed Quotes

```bash
```

```
echo "hello"'world'
echo 'single'"double"'single'
echo "$USER"' is '"$HOME"
```

Unclosed Quotes

```bash
echo "hello
echo 'world
```

4. Environment Variables

Basic Expansion

```bash
echo $USER
echo $HOME
echo $PATH
echo $NONEXISTENT
```

Special Variables

```bash
echo $?
false
echo $?
true
echo $?
exit 42
echo $?  # in new shell
```

Complex Expansion

```bash
echo $USER$HOME
echo "$USER$HOME"
echo '$USER$HOME'
echo test$USERtest
echo $USER_EXTRA  # if USER_EXTRA doesn't exist
```

5. Redirections (Extensive FD Testing)

Basic Output Redirection (>)

```bash
echo hello > file.txt
ls > output.txt
cat file.txt > newfile.txt
> file.txt  # should create/truncate
echo test >file.txt  # no space before filename
echo test> file.txt  # no space after >
echo test>file.txt   # no spaces at all
```

Append Redirection (>>)

```bash
echo hello >> file.txt
echo world >> file.txt
cat file.txt
echo test >>file.txt  # no space
echo test>> file.txt  # no space after >>
echo test>>file.txt   # no spaces at all
```

Input Redirection (<)

```bash
cat < file.txt
wc -l < file.txt
< file.txt cat
cat <file.txt       # no space
cat< file.txt       # no space after cat
cat<file.txt        # no spaces at all
```

File Descriptor Redirections (if supported)

Standard Output (fd 1)

```bash
```

```
echo hello 1>file.txt       # explicit fd 1
echo hello 1> file.txt      # with space after >
echo hello 1 >file.txt      # space after fd number (should fail - parsed as argument)
echo hello 1>file.txt       # no spaces (correct)
ls 1>output.txt
ls 1> output.txt
ls 1 > output.txt           # ambiguous - is 1 an fd or argument?
```

Standard Error (fd 2)

bash

```
ls /nonexistent 2>error.txt  # redirect stderr
ls /nonexistent 2> error.txt
ls /nonexistent 2 >error.txt # space after fd (should fail)
cat nonexist 2>err.txt
grep test file 2>err.txt
ls /fake 2>/dev/null        # common idiom
```

Both stdout and stderr

bash

```
ls /test 1>out.txt 2>err.txt    # separate files
ls /test >out.txt 2>err.txt     # same as above
ls /test 2>err.txt 1>out.txt    # order shouldn't matter
ls /test >out.txt 2>&1          # redirect stderr to stdout (if supported)
ls /test 2>&1 >out.txt          # different behavior
ls /test &>combined.txt         # bash shorthand (if supported)
ls /test >combined.txt 2>&1     # both to same file
```

Input from specific FD (less common)

bash

```
cat 0<file.txt              # explicit fd 0
cat 0< file.txt
cat 0 <file.txt             # space after fd
```

Higher file descriptors (advanced, often not in minishell)

bash

```
echo hello 3>file.txt       # fd 3
echo hello 5>file.txt       # fd 5
cat <&3                     # read from fd 3 (if supported)
exec 3>file.txt             # open fd 3 (if exec supported)
echo test >&3               # write to fd 3
```

Spacing Edge Cases

No spaces

bash

```
cat<input.txt>output.txt
echo hello>file.txt>>file2.txt
ls>a.txt2>b.txt
cat<in.txt|grep test>out.txt
```

Multiple spaces

bash

```
echo hello   >    file.txt
cat    <    input.txt
ls   >>   output.txt
echo test  1>  file.txt  2>  err.txt
```

Tabs

bash

```
echo hello	>	file.txt
cat	<	input.txt
```

Mixed spacing

bash

```
echo hello> file.txt >file2.txt  # different spacing
cat < input.txt>output.txt
ls 2> err.txt>out.txt
```

Ambiguous Cases (Critical Testing)

Is this FD or argument?

```bash
echo 2>file.txt          # redirect fd 2 (stderr)
echo 2 >file.txt         # print "2" then redirect? or fd with space?
echo 1>file.txt          # redirect fd 1 (stdout)
echo 1 >file.txt         # print "1" or redirect?
ls -l 2>errors.txt       # clear: redirect stderr
ls 2 >out.txt            # unclear: is "2" an argument?
```

Multiple digit FDs

```bash
echo hello 10>file.txt     # fd 10 (probably not supported)
echo hello 99>file.txt     # fd 99
echo hello 123>file.txt    # fd 123
```

Invalid FD numbers

```bash
echo hello 2a>file.txt     # not a valid fd
echo hello a2>file.txt     # should be treated as filename
echo hello -1>file.txt     # negative fd
```

Multiple Redirections

Same type, different files

```bash
echo hello >file1.txt >file2.txt        # only file2 gets content
echo hello 1>file1.txt 1>file2.txt      # same behavior
ls 2>err1.txt 2>err2.txt                # only err2 gets errors
```

Different types

```bash
```

```
<input.txt cat >output.txt
cat <in.txt >out.txt 2>err.txt
<in.txt grep test >out.txt 2>err.txt
echo hello >out.txt 2>err.txt <in.txt    # order variation
```

Overwrite vs Append mixing

bash

```
echo hello >file.txt >>file.txt        # what happens?
echo test >>file.txt >file.txt         # different order
```

Heredoc (<<)

Basic heredoc

bash

```
cat <<EOF
hello
world
EOF

cat << EOF                  # space after <<
test
EOF

cat <<EOF                   # no space
test
EOF

cat<< EOF                   # no space before <<
test
EOF
```

Heredoc with FD

bash

```
cat 0<<EOF                    # explicit fd 0
test
EOF

cat 0<< EOF
test
EOF

cat 0 <<EOF                   # space after fd
test
EOF
```

Heredoc with output redirection

bash

```
cat <<EOF >output.txt
content
EOF

cat <<EOF>output.txt          # no space
content
EOF

cat <<EOF 1>output.txt
content
EOF

cat <<EOF 2>errors.txt        # stderr redirect with heredoc
content
EOF
```

Multiple heredocs (if supported)

bash

```
cat <<EOF1 <<EOF2                    # second one wins? error?
first
EOF1
second
EOF2

cat <<EOF >>output.txt
append this
EOF
```

Heredoc delimiters

bash

```
cat <<EOF
test
EOF

cat <<'EOF'                          # quoted delimiter (no expansion)
$USER
EOF

cat <<"EOF"                          # double quoted delimiter
$USER
EOF

cat <<E                              # single char delimiter
test
E

cat <<"                              # empty delimiter (weird)
test
"
```

Invalid/Error Cases

Missing filename

bash

```
echo hello >
echo hello >>
cat <
ls 2>
echo test 1>
```

Invalid filenames

bash

```
echo hello >/invalid/deep/path/file.txt
cat </nonexistent/file.txt
echo test >/dev/invalid
ls 2>/proc/invalid
```

Permission issues

bash

```
echo hello >/root/file.txt          # permission denied
cat </etc/shadow                    # permission denied
echo test >/readonly/file.txt       # readonly filesystem
```

Conflicting redirections

bash

```
cat <file1.txt <file2.txt           # which one?
echo hello 1>out.txt 1>out2.txt     # last one wins
```

Unclosed heredoc

bash

```
cat <<EOF
this is not closed
# no EOF - should wait or error
```

Invalid syntax

bash

```
echo > > file.txt                # double >
cat < < file.txt                 # double <
echo >> > file.txt               # mixed operators
ls >                             # no filename
cat 2 2>errors.txt               # ambiguous
```

Special Files

/dev/null

bash

```
echo hello >/dev/null
ls /fake 2>/dev/null
cat file.txt >/dev/null 2>&1
```

/dev/urandom, /dev/zero

bash

```
cat </dev/urandom | head -c 10
cat </dev/zero | head -c 10
```

Directories

bash

```
echo hello >./                   # should fail
cat <./                          # should fail
ls >directory_name               # should fail
```

Redirection with Pipes

Before pipe

bash

```
cat <input.txt | grep test
echo hello >file.txt | cat       # does this make sense?
ls 2>err.txt | grep test
```

After pipe

```bash
cat file.txt | grep test >output.txt
ls | cat >output.txt 2>errors.txt
echo hello | cat | cat >final.txt
```

Both sides

```bash
cat <input.txt | grep test >output.txt
<in.txt cat | grep test >out.txt 2>err.txt
```

Complex Real-World Scenarios

```bash
# Compile and redirect stderr
gcc test.c 2>errors.txt

# Save both stdout and stderr separately
make 1>build.log 2>errors.log

# Discard errors
ls /fake /real 2>/dev/null

# Append logs with timestamps
echo "[$(date)] Log entry" >>app.log 2>>error.log

# Complex pipeline with multiple redirects
cat <input.txt | grep "error" 2>grep_err.txt | sort >sorted.txt

# Multiple files with heredoc
cat <<EOF >file1.txt && cat <<EOF2 >file2.txt
content1
EOF
content2
EOF2

# Redirect in subshell (if supported)
(echo hello >inner.txt)
cat inner.txt
```

Boundary Testing

```bash
# Very long filenames
echo hello >very_long_filename_that_goes_on_and_on_and_on.txt

# Special characters in filenames (if supported)
echo test >"file with spaces.txt"
echo test >'file$with$dollars.txt'
echo test >file\$escaped.txt

# Numeric-looking filenames
echo hello >123.txt
echo hello >456
cat <789

# FD-like filenames
echo hello >1.txt                # file named "1.txt"
echo hello >2.txt                # file named "2.txt"
cat <0.txt                       # file named "0.txt"
```

Stress Tests

```bash
# Many redirections
echo hello >1.txt >2.txt >3.txt >4.txt >5.txt

# Deep redirection nesting with pipes
cat <in.txt | grep a >tmp1.txt && cat <tmp1.txt | grep b >tmp2.txt

# Large heredoc
cat <<EOF >large.txt
[1000 lines of text]
EOF
```

6. Pipes

Simple Pipes

```bash
```

```
ls | grep test
cat file.txt | wc -l
echo hello | cat
```

Multiple Pipes

```bash
ls -l | grep test | wc -l
cat file.txt | grep hello | sort | uniq
```

Pipes with Redirections

```bash
cat < input.txt | grep test > output.txt
ls | cat > files.txt
< input.txt cat | grep test
```

Empty Pipes

```bash
| cat
cat |
cat | | cat
```

7. Logical Operators

AND (&&)

```bash
true && echo success
false && echo should not print
ls && pwd
cd /tmp && pwd
```

OR (||)

```bash
```

```
true || echo should not print
false || echo failure
ls nonexistent || echo file not found
cd /invalid || echo cannot change directory
```

Combined

bash

```
true && echo first || echo second
false && echo first || echo second
true || echo first && echo second
```

8. Subshells and Parentheses

Basic Parentheses

bash

```
(echo hello)
(cd /tmp && pwd)
echo $PWD  # should still be in original directory
```

Nested Parentheses

bash

```
(echo outer; (echo inner))
((echo nested))
```

9. Wildcards (if implemented)

bash

```
ls *.txt
echo *
cat file*.txt
ls *.c *.h
```

10. Signal Handling

Ctrl+C (SIGINT)

bash

```
cat  # press Ctrl+C
sleep 100  # press Ctrl+C
cat | cat | cat  # press Ctrl+C
```

Ctrl+D (EOF)

```
bash

cat  # press Ctrl+D
# press Ctrl+D on empty prompt (should exit)
```

Ctrl+\ (SIGQUIT)

```
bash

cat  # press Ctrl+\ (should do nothing in interactive mode)
```

11. Edge Cases

Empty Input

```
bash

# just press Enter


  # spaces only
   # tabs only
```

Whitespace Handling

```
bash

  echo    hello   world
echo  "tabs  here"
```

Special Characters

```
bash

echo ;;;
echo |||
echo &&&
echo <<<
echo >>>
```

Long Commands

```bash
echo 1234567890123456789012345678901234567890123456789012345678901234567890...
```

PATH Issues

```bash
unset PATH
ls  # should fail or use absolute path
export PATH=/bin:/usr/bin
ls  # should work again
```

Multiple Semicolons (if supported)

```bash
echo hello ; echo world
echo 1; echo 2; echo 3
; echo test  # invalid
echo test ;  # valid
```

12. Exit Status

```bash
true
echo $?  # should be 0

false
echo $?  # should be 1

ls nonexistent
echo $?  # should be non-zero

ls && echo $?  # should be 0
ls || echo $?  # should not execute second part

exit 42
# in new shell:
echo $?  # should be 42
```

13. Memory and Leak Testing

Run with valgrind:

```bash
valgrind --leak-check=full --show-leak-kinds=all ./minishell
```

Test commands:

```bash
# Run many commands
ls
pwd
echo hello
env
export TEST=value
unset TEST
exit

# Long running operations
cat | cat | cat | cat
# Ctrl+C

# Many pipes
ls | cat | cat | cat | cat | cat | wc -l
```

14. Stress Tests

Many Arguments

```bash
echo 1 2 3 4 5 6 7 8 9 10 ... 100
```

Deep Pipes

```bash
cat file | cat | cat | cat | cat | cat | cat | cat
```

Many Environment Variables

```bash
```

```
export VAR1=1 VAR2=2 VAR3=3 ... VAR100=100
env
```

Large Heredoc

```bash
cat << EOF
[many lines of text]
EOF
```

15. Comparison with Bash

For any test, compare output with bash:

```bash
# In bash:
<command> > bash_output.txt 2>&1
echo $? >> bash_output.txt

# In your minishell:
<command> > minishell_output.txt 2>&1
echo $? >> minishell_output.txt

# Compare:
diff bash_output.txt minishell_output.txt
```

Common Pitfalls to Test

1. **Unclosed quotes in pipes**: `echo "hello | cat`
2. **Multiple redirections to same fd**: `echo hi > a > b > c`
3. **Mixing redirections and heredoc**: `cat << EOF > file.txt`
4. **Empty environment variable**: `echo $EMPTY`
5. **Dollar sign without variable**: `echo $ test`
6. **Export with invalid names**: `export 1VAR=test`
7. **CD with multiple arguments**: `cd /tmp /var`
8. **Exit with non-numeric argument**: `exit abc`

9. **Pipe at beginning or end**: `| cat` or `cat |`
10. **Consecutive operators**: `echo && && pwd`