VE482 Project 1 Grading Policy:

Overview

This document lists grading scheme used for grading your project1. A copy of source code of this document and the test programs used are available at

```
git clone git@ve482:p1
```

The general procedure of grading your project 1 include:

- **You declare which subset of all the features are supported by your program**.
- We compile your program on both Linux and Minix. Your program are required to compile both on Minix 3.3.0 and any properly setup common Linux distribution
- We decide on a list of commands that will be executed on your work. We pick the commands to execute according your declared supported features.
- We execute them one by one on **Linux** only (we assume that your program behavior is consistent across platforms. Your program behavior will be compared to the behavior of `sh` program available in most Unix systems.

Your program should conform to standard C (compiles with `-pedantic` nd `-Werror` options enabled). Your are free to choose the language standard version, but do please specify them in your **README**. We will compile your program under C11 standard if not specified. Your program should work properly with second level optimization enabled (`-02` option).

If your program fails to compile on either Linux or Minix, a 10% deduction will be applied. If your program does not compile on any of the systems, we will try to fix this for you (best effort, no guarantees). A 20% deduction will be applied. If your program still cannot compile, we will invite you for a cup of tea in our office hours.

It is very important that you declare your supported subset of features. We will not grade your program until we have received your declaration. You can only declare a feature as supported if you have declared all of it's dependent feature as supported. You can find a template for you in Appendix A section of this document. We will only grade on the features that you report as "supported". You should include this declaration in your **README** file. A sample is declaration is provided as follows:

```
[x] 1. Write a working read/parse/execute loop and an exit command; [5]
[x] 2. Clean exit, no memory leaks in any circumstance; [5]
[x] 3. Handle single commands without arguments (e.g. ls); [5]
[x] 4. Support commands with arguments (e.g. apt-get update or pkgin update); [5]
5. File I/O redirection: [5+5+5+2]
[x] 5.1. Output redirection by overwriting a file (e.g. echo 123 > 1.txt);
[x] 5.2. Output redirection by appending to a file (e.g. echo 465 >> 1.txt);
[x] 5.3. Input redirection (e.g. cat < 1.txt);
[ ] 5.4. Combine 5.1 and 5.2 with 5.3;
[ ] 6. Support for bash style redirection syntax (e.g. cat < 1.txt 2.txt > 3.txt 4.txt); [8]
7. Pipes: [5+5+5+10]
[x] 7.1. Basic pipe support (e.g. echo 123 j grep 1);
[x] 7.2. Run all ‘stages’ of piped process in parallel. (e.g. yes ve482 j grep 482);
[x] 7.3. Extend 7.2 to support requirements 5 and 6 (e.g. cat < 1.txt 2.txt j grep 1 > 3.txt);
[ ] 7.4. Extend 7.3 to support arbitrarily deep \cascade pipes" (e.g. echo 123 j grep 1 j grep 1
j grep 1)
Note: the sub-processes must be reaped in order to be awarded the marks.
[x] 8. Support CTRL-D (similar to bash, when there is no/an unfinished command); [5]
9. Internal commands: [5+5+5]
[x] 9.1. Implement pwd as a built-in command;
[x] 9.2. Allow changing working directory using cd;
[ ] 9.3. Allow pwd to be piped or redirected as specified in requirement 5;
10. Support CTRL-C: [5+3+2+10]
[x] 10.1. Properly handle CTRL-C in the case of requirement 5;
[x] 10.2. Extend 10.1 to support subtasks 7.1 to 7.3;
[x] 10.3. Extend 10.2 to support requirement 8, especially on an incomplete input;
[ ] 10.4. Extend 10.3 to support requirement 7;
11. Support quotes: [5+2+3+5]
[x] 11.1. Handle single and double quotes (e.g. echo "de'f' ghi" '123"a"bc' a b c);
[x] 11.2. Extend 11.1 to support requirement 5 and subtasks 7.1 to 7.3;
[ ] 11.3. Extend 11.2 in the case of incomplete quotes (e.g. Input echo "de, hit enter and input
cd");
[x] 11.4. Extend 11.3 to support requirements 5 and 7, together with subtask 10.3;
12. Wait for the command to be completed when encountering >, <, or j: [3+2]
[ ] 12.1. Support requirements 4 and 5 together with subtasks 7.1 to 7.3;
[ ] 12.2. Extend 12.1 to support requirement 11
```

For each time your program crashes (or hangs) during the grading process. we will apply a 3% deduction to your final score. Total deduction is capped at 30%. We wil not rerun the command causing the crash after we restart your program.

Any failed command under a feature will cost you **all** points related to that feature (or sub-feature if applicable), however a feature may still be considered "completed" (so that other features depending on it may still be graded). A feature is considered "completed" only if reasonable degree of support is provided. E.g. supporting ONLY `ls` does not count as supporting requirement 3. A feature is grade only if all it's dependant features are graded as supported.

We have created a few test programs to facilitate the testing of your shell. Their source code are available in the public readable `git` given above. We refer them as `tprog1`, `tprog2` ... in the following test cases.

The test cases

1. **Write a working read/parse/execute loop and an exit command; [5]**

- Graded by reviewing your code

2. **Clean exit, no memory leaks in any circumstance; [5]**

- By running your program through `valgrind` and carefully read the output.

3. **Handle single commands without arguments (e.g. ls); [5]**

- Execute `ls` , `man` and `info` (exit by hitting `q`)
- Execute `nano` and press `ctrl+x`

4. **Support commands with arguments (e.g. apt-get update or pkgin update);**

- Execute `pkgin update` (or `sudo apt-get update` on Linux)
- Execute `echo 123 abc` and `ls -al`
- Execute command with more than 300 arguments.
- Exectue `tprog1`

5. **File I/O redirection: [5+5+5+2]**
   5.1. **Output redirection by overwriting a file (e.g. echo 123 > 1.txt);**

- Execute `echo 123 > 1.txt`
- Execute `head /dev/urandom > 1.txt`
- Execute `tprog1 > 1.txt`
  5.2. **Output redirection by appending to a file (e.g. echo 465 >> 1.txt);**
- Execute `head /etc/passwd >> 1.txt` twice.
- Execute `head /etc/passwd >> 2.txt`
- Execute `tprog1 >> 1.txt`
  5.3. **Input redirection (e.g. cat < 1.txt);**
- Execute `head < 1.txt`
- Execute `diff -y - 1.txt < 1.txt`
- Execute assuming `t.in` contains a string `test` , then `tprog < t.in`
  5.4. **Combine 5.1 and 5.2 with 5.3;**
- Execute `diff -y - 1.txt < /dev/urandom > 1.txt`
- Execute `tprog1 >> 1.txt < t.in`

6. **Support for bash style redirection syntax (e.g. cat < 1.txt 2.txt > 3.txt 4.txt); [8]**

- Execute `diff -y - 1.txt</dev/urandom>1.txt`
- Execute `>>1.txt diff -y 1.txt -</dev/urandom 2.txt`
- Execute `</dev/urandom time head -10>>1.txt 1.txt`
- Execute `tprog1<test.in >> /dev/null 2.txt 3.txt`
- Execute `time tprog1>>/dev/null 2 3 4 < test.in 4 5 6`

7. **Pipes: [5+5+5+10]**
   7.1.**Basic pipe support**

- Execute `echo test | tprog1`
- Execute `find /etc/init -type f | xargs cat`
- After each command check if the sub process has terminated
  7.2. **Run all 'stages' of piped process in parallel.**
- Execute `yes ve482 | yes abc | head -10`

- Execute `echo test | tprog1 | tprog1`
- Execute `yes ve482 is simple | yes ve477 is easy | yes lets meet at usc | head -10`

7.3. **Extend 7.2 to support requirements 5 and 6**

- Execute `tprog1<1.txt abc|tee a.txt |tprog1|>/dev/null tee b.txt`
- Execute `time head -10</dev/urandom | tprog1>> /dev/null`
- Execute `</dev/urandom time head -10 |>/dev/null tprog1`
- Execute `yes|yes|grep y|diff ->/dev/null 1.txt|echo abc`

7.4. **Extend 7.3 to support arbitrarily deep "cascaded pipes"**

- One should expect hundreds of piped process.
- Execute a long command with cascaded execution of `yes` : e.g. `yes|...|yes|head -10`
- Execute a long command with cascased execution of `tprog2` : e.g. `tprog2|tprog2|....`
- Execute a long command with cascased execution of `tprog2` ended with `echo abc`
- Execute a long command with cascased execution of `tprog3` ended with `echo abc`
- Check with `ps -ax` to make sure all sub-processes are terminated.

8. **Support CTRL-D (similar to bash, when there is no/an unfinished command); [5]**

- This feature will be tested after all other features are tested.
- Type anything and hit CTRL-D. Your shell should do nothing.
- Execute `head -1000 /dev/urandom` and hit CTRL-D during execution.
- Type `echo ve482` and hit CTRL-D mutiple times. then hit ENTER
- Clear the content and hit CTRL-D, your shell should exit.

9. **Internal commands: [5+5+5]**

9.1. **Implement `pwd` as a built-in command**

- Read your implementation of `pwd` code
- Execute `pwd` and see what happens

9.2. **Allow changing working directory using `cd`**

- Execute `pwd` , then execute `cd ..` followed by `cd .` and another `pwd`
- Execute `cd /etc/../etc/./././../etc`
- Execute `cd` alone.
- Execute `cd` with more than 1 argument

9.3. **Allow pwd to be piped or redirected as specified in requirement 5**

- Execute `pwd < 1.txt | diff -y 1.txt - | pwd | tee pwd.tee >> /dev/null`

10. **Support CTRL-C: [5+3+2+10]**

10.1. **Properly handle CTRL-C in the case of requirement 5;**

- Execute `cat /dev/urandom` and hit CTRL-C in the process
- Execute `cat /dev/urandom < /dev/urandom > /dev/null` and hit CTRL-C during execution
- Execute `tprog1 ve482 >> 1.txt` and hit CTRL-C during execution.

10.2. **Extend 10.1 to support subtasks 7.1 to 7.3;**

- Execute `tprog1 abc|tee a.txt |tprog1|>/dev/null tee b.txt` and hit CTRL-C
- Execute `time yes ve482|grep ve|>> /dev/null grep 482` and hit CTRL-C
- Execute `yes|yes|grep y|diff ->/dev/null 1.txt` and hit CTRL-C

10.3. **Extend 10.2 to support requirement 8, especially on an incomplete input;**

- This feature is tested before feature 8.
- Type nothing and hit CTRL-C multiple times.
- Type `yes ve482` and hit CTRL-D mutiple times. then hit ENTER, then CTRL-C

- Type random content and hit CTRL-C multiple times

10.4. **Extend 10.3 to support requirement 7;**

- One should expect hundreds of piped process.
- Execute a long command with cascaded execution of `yes` : e.g. `yes|...|yes` and hit CTRL-C
- Execute a long command with cascased execution of `tprog2` : e.g. `tprog2|tprog2|....` and hit CTRL-C
- Check with `ps -ax` to make sure all sub-processes are terminated.

11. **Support quotes: [5+2+3+5]**

11.1.**Handle single and double quotes**

- Execute `"echo" 'abc def' "txm""xtt"`
- Execute `'tprog4' "abc' ''abc" "'" "abc" '"'`

11.2. **Extend 11.1 to support requirement 5 and subtasks 7.1 to 7.3**

- Execute `"echo" "<1.'txt'" < 1.txt > "2.""txt"`
- Execute `tprog4 " abc'" <'1.txt' >> 2."'txt'"`
- Execute `tprog4 " abc'" | grep ' ' | tee 2.''' >> 2."'txt'"`

11.3. **Extend 11.2 in the case of incomplete quotes**

- Type `tprog4 "` then hit ENTER multiple times and type `abc"`
- Type `'` and hit ENTER, continues with `echo' abc"
- Type `'tprog4' "<1.'txt'` , hit ENTER, and type `" < 1.txt >> 2."'txt'"`
- Breakup commands used in testing feature 11.2 at arbitrary locations with newline character

11.4. **Extend 11.3 to support requirements 5 and 7, together with subtask 10.3**

- Use cases in 11.2 and hit CTRL-C at arbitrary location (especailly with after pipes/redirects)

12. **Wait for the command to be completed when encountering `>` , `<` , or `|` : [3+2]**

12.1. **Support requirements 4 and 5 together with subtasks 7.1 to 7.3**

- Abitrarily break up commands used in 7.1 through 7.3 at `>` , `<` or `|`

12.2. **Extend 12.1 to support requirement 11**

- Abitrarily break up commands used in 11 at `>` , `<` or `|`

13. **Errors to be detected (each error missed costs 2 pts)**:

- Be advised an error is considered caught only if it is caught in **all** supported features
- **Also be advised we will not test for not supported feature**. If your program does not support pipes, we will non expect you to caught errors involving pipes.
- Nonexisting program e.g. `non-exist abc def` and `echo abc | non-exsiting`
- Nonexisting file in input redirection e.g. `cat < non-existing.txt`
- Failed to open file in output redirection e.g. `echo abc > /dev/permission_denied`
- Duplicated input redirection e.g. `echo abc < a.txt < b.txt`
- Duplicated output redirection e.g. `echo abc > a.txt >> b.txt` , `echo abc > a.txt | grep abc`
- Syntax Error e.g. `echo abc > > > >` , `echo abc > < 1.txt` , `echo abc > | grep abc` etc.
- Missing program e.g. `> abc | | grep 123`
- `cd` to non-existing directory e.g. `cd /tan90/`

Appendix A

[ ] 1. Write a working read/parse/execute loop and an exit command; [5]
[ ] 2. Clean exit, no memory leaks in any circumstance; [5]
[ ] 3. Handle single commands without arguments (e.g. ls); [5]
[ ] 4. Support commands with arguments (e.g. apt-get update or pkgin update); [5]
5. File I/O redirection: [5+5+5+2]
[ ] 5.1. Output redirection by overwriting a file (e.g. echo 123 > 1.txt);
[ ] 5.2. Output redirection by appending to a file (e.g. echo 465 >> 1.txt);
[ ] 5.3. Input redirection (e.g. cat < 1.txt);
[ ] 5.4. Combine 5.1 and 5.2 with 5.3;
[ ] 6. Support for bash style redirection syntax (e.g. cat < 1.txt 2.txt > 3.txt 4.txt); [8]
7. Pipes: [5+5+5+10]
[ ] 7.1. Basic pipe support (e.g. echo 123 j grep 1);
[ ] 7.2. Run all ‘stages’ of piped process in parallel. (e.g. yes ve482 j grep 482);
[ ] 7.3. Extend 7.2 to support requirements 5 and 6 (e.g. cat < 1.txt 2.txt j grep 1 > 3.txt);
[ ] 7.4. Extend 7.3 to support arbitrarily deep \cascade pipes" (e.g. echo 123 j grep 1 j grep 1 j grep 1)
Note: the sub-processes must be reaped in order to be awarded the marks.
[ ] 8. Support CTRL-D (similar to bash, when there is no/an unfinished command); [5]
9. Internal commands: [5+5+5]
[ ] 9.1. Implement pwd as a built-in command;
[ ] 9.2. Allow changing working directory using cd;
[ ] 9.3. Allow pwd to be piped or redirected as specified in requirement 5;
10. Support CTRL-C: [5+3+2+10]
[ ] 10.1. Properly handle CTRL-C in the case of requirement 5;
[ ] 10.2. Extend 10.1 to support subtasks 7.1 to 7.3;
[ ] 10.3. Extend 10.2 to support requirement 8, especially on an incomplete input;
[ ] 10.4. Extend 10.3 to support requirement 7;
11. Support quotes: [5+2+3+5]
[ ] 11.1. Handle single and double quotes (e.g. echo "de'f' ghi" '123"a"bc' a b c);
[ ] 11.2. Extend 11.1 to support requirement 5 and subtasks 7.1 to 7.3;
[ ] 11.3. Extend 11.2 in the case of incomplete quotes (e.g. Input echo "de, hit enter and input cd");
[ ] 11.4. Extend 11.3 to support requirements 5 and 7, together with subtask 10.3;
12. Wait for the command to be completed when encountering >, <, or j: [3+2]
[ ] 12.1. Support requirements 4 and 5 together with subtasks 7.1 to 7.3;
[ ] 12.2. Extend 12.1 to support requirement 11