Group Members (ask who participated):

Stages attempted:

Run all the tests from a linux terminal preferably on a lab machine!

Stage 1

Is lack of input handled correctly (just hitting <enter> should print again the prompt)?

Does the shell exit when you type ‘exit’?

Does the shell exit when you type <ctrl>D?

Try both the above as the first thing after loading the shell:

> Are any rubbish printed?
>
> Is there a segmentation fault?

Is the input tokenized correctly (try ‘ls<tab>-lF;.&..>.<..|/<tab>fksdk’ it should print an error for fksdk and the listings of . and .. twice plus the listing of / all formatted with entry per line and file permissions)?

> Are additional characters beyond (space \t \n ; & > < |) used in the code for tokenization? They shouldn’t be.

Stage 2

Are external commands executed correctly (the above input line tests this, try also ‘ps aux’, and ‘clear’)?

Are non-existing external commands handled correctly:

> Is error message printed?
>
> Does the error message contain the name of attempted command? Does it use perror for the message?
>
> Is the child process exited (with one ctrl<D> after the command the shell should completely exit)?

Stage 3

Does the shell start in the user’s home directory (check with pwd)?

Does getpath print the current path?

Does setpath /bin set the current path to /bin (check with getpath and also executing commands I think ls should still work but clear not)?

Do they restore the path on exit and print the path after restoration:

Do they print the getenv return value for the path on exit?

Do they restore and print the path on both ways of exiting (exit and <ctrl>-D)?

Do they print appropriate error messages:

When getpath has parameters?

When setpath has no parameters?

When setpath has more than 1 parameter?

Do the error messages say what the problem is and how to fix it?

Stage 4

Does cd <directory> change the current directory correctly (try with absolute e.g. /bin and relative e.g. directories ../<username> or from /bin something like .././bin/./.. (should end up in /) check with pwd and ls)?

Does cd get you back to the user's home (check with pwd and ls)?

Are cd errors dealt with correctly:

Does cd to a non-existent directory or file print 'no such file or directory'?

Does cd to an existing file print 'not a directory'?

Do the error messages contain the name of the file that caused the error?

Does cd with 2 or more parameters print an error message?

Does the error message say what the problem was and how to fix it?

Stage 5

<u>Start testing history by deleting any history from the history file (i.e. with clear history)!</u>

Does history print the contents of the history correctly:

Does every command in the history also include its parameters?

Are all commands in history listed with a number in front of them?

Are all the numbers in ascending order?

Do the numbers in the list start from 1(not 0)?

Is the last command at the bottom of the list and with the highest number?

Does typing history print a list with history as the last command?

Are any empty history positions printed (i.e. numbers with no commands)? They shouldn't be

Do the above also apply when the history is full (typically after 20 entries (ask about the size of their history for that))?

Are history invocations with **positive** numbers (i.e. !<no>) handled correctly:

Is the invoked command including its parameters executed (try ls –lF and then invoke it from history)?

Do invocations for both internal and external commands work correctly?

Can you make the same invocation multiple times (note the invocation shouldn't change how the command shows in history)?

Do history invocations leave the history unchanged (invoking the history command from the history should print exactly the same listing)?

Are history invocations with **negative** numbers (i.e. !-<no>) handled correctly:

Is the invoked command including its parameters executed (try ls –lF and then invoke it from history)?

Do invocations for both internal and external commands work correctly?

Can you make the same invocation multiple times?

Is the correct command invoked (i.e. -1 last command, -2 one before it, etc)?

Do history invocations leave the history unchanged (invoking the history command from the history should print exactly the same listing)?

Is history error handling correct:

Is error message printed when a history invocation is attempted as the first command with clear history?

Is error message printed when number greater than the number of commands in history but less than the maximum is given?

Is error message printed when a number beyond the maximum is used in a history invocation?

Is an error message printed when a negative number that is out of range is used in a history invocation?

Does history with parameters print an error message?

Are error messages provided when ! is followed by something different than a positive or negative number?

Are error messages provided when !!, !<no> or !-<no> are followed by something else? Note that it's fine if they allow extra things but these are added as extra parameters.

Are all error messages clear of what went wrong?

Stage 6

Does the file containing the saved history contain the right information (commands in the correct order)? Is this also the case when the history is full (more than the limit of commands was provided)?

Is the history file called .hist_list?

Is the history file located in the user's home?

Can you see the correct history listing when history is the first command (i.e. the commands from the file with history added as the last command (note it may replace the first if the history was full))?

Can you invoke a command from the saved history as the first command?

Is the error handling correct:

Does the shell crash if you start it from a different directory?

Does the shell crash if you start it after deleting the history file?

Does the shell crash if you change to / and then exit (i.e. trying to save the file in the current directory rather than the users home)?

Does the shell crash if the history file exists but is empty (you can use touch to create an empty history file)?

Does the shell crash if the history file contents are random (particularly relevant if they include numbers in the file)?

Stage 7

Does the alias command give appropriate message when the list is empty (no aliases set yet)?

Are aliases correctly set including any command parameters (check by calling alias to see the correct listing)?

Can you execute aliases correctly:

> Does the execution take into account the command parameters?
>
> Can you execute the alias multiple times?
>
> Can you execute it with additional parameters (i.e. alias la to execute ls –la and then call la / which should execute ls –la /)?
>
> Can you use internal commands as aliases (e.g. alias cd to execute ps)?
>
> Can you alias internal commands (e.g. alias bin to execute cd /bin)?
>
> Are aliases listed in the history (the alias should appear not the command that has been aliased)?
>
> Can you invoke an alias from the history (should execute the aliased command)?
>
> Are parameters handled correctly when aliases are invoked from the history (e.g. alias la to execute ls –la then call la / followed by history and then call !-2 that should execute ls –la /)?

Is there a check for existing aliases when an alias is defined (either an error message or a message for overriding)?

If overriding aliases, is this done correctly (i.e. alias p to execute ps aux, then alias p to execute ps, run p that should execute just ps)?

Can you unalias aliases correctly:

> Unalias when the list is not full removes the alias (check that it does execute any more)?
>
> Unalias when the list is full creates an opening (i.e. you can introduce a new alias)?

Is the error handling correct:

> Does alias with 1 parameter give an error message?
>
> Does an attempt to alias when the list is full, give an error message?
>
> If using aliasing an existing alias overrides the alias, then does overriding also work when the list is full (i.e. no error message, just the overriding message)?

Does attempting to unalias when the list of aliases is empty give an error message?

Does attempting to unalias a non-existing alias give an error message?

Are the messages always clear on what the problem is?

Stage 8

Does the file containing the saved aliases contain the right information (alias and aliased commands in the correct order)? Is this also the case when aliases are full (the limit of aliased commands supported)?

Is the aliases file called .aliases?

Is the aliases file located in the user's home?

Can you see the correct aliases listed when alias is the first command (i.e. the aliases commands from the file)?

Can you invoke an alias from the saved aliases as the first command?

Can you invoke a command from the saved history that refers to an alias with the aliased command executing? Can you do the same for an alias followed by parameters?

Is the error handling correct:

Does the shell crash if you start it from a different directory?

Does the shell crash if you start it after deleting the aliases file?

Does the shell crash if you change to / and then exit (i.e. trying to save the file in the current directory rather than the users home)?

Does the shell crash if the aliases file exists but is empty (you can use touch to create an empty aliases file)?

Does the shell crash if the aliases file contains rubbish (do they check for the format of the file contents)?

Stage 9

Can you set alias chains that the first alias would execute the last in the chain?

Can you alias a history invocation correctly (i.e. alias five !5 or alias minus !-1)?

Do aliases on history invocations change as the history updates (e.g. does minus above always execute the last command in history)?

Are alias cycles detected either at execution (e.g. alias a b then alias b a then executing a should give an error) or at definition (e.g. alias a b the alias b a should give an error)?

Are the error messages clear on what the problem is?

Code Quality

Is the indentation appropriate and consistent throughout?

Are all functions declared reasonable (i.e. have a clear single purpose, include all code related to their purpose (e.g. parameter checking for built-in commands is part of the function that implements the command) with not too much code)?

Are the comments appropriate (the purpose of each function is described, they are succinct and not extensive, the comments inside functions really aid understanding)?

Are there any global variables?

If there are, how many?

Are there any #define that include code not just constant values?

Is the code split into multiple files?

If it is, is each file reasonable (i.e. has a clear purpose, includes all code related to that purpose, and it doesn’t contain too little or too much code)?

Are the any variables that files expose to the rest of the program (i.e. indirectly global variables)?