

(https://profile.intra.42.fr)

SCALE FOR PROJECT 42SH (/PROJECTS/42SH)

You should evaluate 5 students in this team



Git repository

vogsphere@vogsphere.13 

Introduction

Nous vous demandons pour le bon déroulement de cette évaluation de respecter les règles suivantes :

Vous utilisez le SHELL depuis de nombreux mois, vous savez comment il doit réagir.

Soyez imaginatif sur les tests d'erreur et souvenez vous que trouver un SEGFAULT signifie pour les corrigés que leur travail n'a pas été assez minutieux. Pour le correcteur, cela signifie qu'il doit s'assurer de discuter de ce qui provoque ce SEGFAULT avec ses corrigés.

En cas de segfault, continuez néanmoins la soutenance pour échanger sur le sujet (sans appliquer le barème, puisque vous aurez bien entendu sélectionné l'onglet CRASH ci dessous).

- Acceptez qu'il puisse y avoir parfois des différences d'interprétation sur les demandes du sujet ou l'étendue des fonctionnalités. Restez ouvert d'esprit face à la vision de l'autre (a-t-il ou elle raison ou tort ?), et notez le plus honnêtement possible. La pédagogie de 42 n'a de sens que si la peer-évaluation est faite sérieusement.

Guidelines

- Vous ne devez évaluer que ce qui se trouve sur le dépôt GiT de rendu de l'étudiant(e) ou du groupe.

Corrigez d'abord la partie obligatoire. Si et seulement si cette partie obligatoire est PARFAITE (complète et indestructible !),

passer à la partie modulaire. Gardez à l'esprit que le shell doit être intuitif, mais que les corrigés restent libres de l'implémentation de certains détails. Leur shell de référence n'est pas forcément le même que le votre.

La correction peut être longue si la partie modulaire est bien fournie. Le 42sh n'est pas un projet trivial, prenez le temps qu'il faut pour examiner minutieusement le travail de vos camarades.

- Tout script sensé faciliter l'évaluation fourni par l'un des deux partis doit être rigoureusement vérifié par l'autre parti pour éviter des mauvaises surprises.

- Si l'étudiant(e) correcteur/correctrice n'a pas encore fait ce projet, il est obligatoire pour cet(te) étudiant(e) de lire le sujet en entier avant de commencer cette soutenance.

- Utilisez les flags disponibles sur ce barème pour signaler un rendu vide, non fonctionnel, une faute de norme, un cas de triche, etc. Dans ce cas, l'évaluation est terminée et la note finale est 0 (ou -42 dans le cas spécial de la triche). Toutefois, hors cas de triche, vous êtes encouragés à continuer d'échanger autour du travail effectué (ou non effectué justement) pour identifier les problèmes ayant entraîné cette situation et les éviter pour le prochain rendu.

- La valeur de retour peut être récupérée avec "echo \${?}".


- Le prompt de l'utilisateur sera représenté par "\$>" dans tous les tests du barème. Vous devrez donc adapter votre correction en fonction du prompt du groupe évalué.

- Les parties contenues dans les placeholders "{ { ... } }" indiquent une partie qui est susceptible de varier selon les corrections.

A la place, nous vous indiquons ce qui est susceptible de s'afficher.

De ce fait, il est de votre responsabilité de vérifier que la partie affichée soit bien en adéquation avec le thème du placeholder.

Attachments

 files needed for correction (/uploads/document/document/1549/42sh.txts.tgz)

 Subject (<https://cdn.intra.42.fr/pdf/pdf/6352/42sh.fr.pdf>)

Partie obligatoire

Rappel : si à un moment ou un autre, le programme ne réagit pas correctement (bus error, segfault, etc...), ou bien si vous détectez une fuite mémoire, la soutenance est terminée et la note est 0. Pensez à utiliser les flags correspondants quand cela est nécessaire. Cette consigne est active d'un bout à l'autre de la soutenance.

Fichier auteur

Vérifiez d'abord les éléments suivants :

- Il y a bien un rendu (dans le dépôt git)
- Fichier auteur valide
- Le Makefile est présent et compile bien l'exécutable 42sh
- Pas de faute de norme, la Norminette faisant foi
- Pas de triche (fonctions non autorisées, les étudiants doivent pouvoir expliquer leur code, ...)

Si un élément n'est pas conforme au sujet, la notation s'arrête là. Vous êtes encouragés à continuer de débattre du projet, mais le barème n'est pas appliqué.

☒ Yes

☐ No

Fuites mémoire

Pendant toute la durée de la soutenance, gardez un oeil sur les possibles fuites mémoire du 42sh (via cette commande dans un autre terminal par exemple "while true; do leaks 42sh; sleep 1; clear; done". Si leaks rapporte une fuite mémoire, la note du projet est 0.

☒ Yes

☐ No

Prérequis du minishell

Nous allons évaluer les prérequis du minishell. Si l'un au moins de ces tests échoue, alors toute la section est échouée et la correction s'arrête. Effectuez les tests suivants:

- Exécutez une commande vide "\$> ". Le shell ne doit rien faire et réafficher le prompt.
- Exécutez une commande composée uniquement d'un seul espace "\$> ". Le shell ne doit rien faire et réafficher le prompt.
- Exécutez une commande composée uniquement d'espaces et de tabulations. Le shell ne doit rien faire et réafficher le prompt.
- Exécutez une commande avec plusieurs espaces et tabulations avant le nom du binaire, entre chaque argument passé au

binaires, et après le dernier argument. Ces espaces et tabulations inutiles ne doivent pas perturber l'exécution de la commande.

- Testez une commande (et/ou un built-in) avec des options inexistantes. Vérifiez que le retour de la commande n'est pas 0.

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> doesnotexist
{{ Message d'erreur indiquant que la commande n'existe pas/est introuvable }}
$> echo ${?}
{{ Code de retour différent de 0 }}
```

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> /sbin/yubikey_shell
{{ Message d'erreur indiquant que la commande ne peut être exécuté pour cause de permission insuffisante }}
$> echo ${?}
{{ Code de retour différent de 0 }}
```

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> /bin/ls
{{ Sortie de la commande "ls" }}
$> echo ${?}
0
```

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> ls
{{ Sortie de la commande "ls" }}
$> echo ${?}
0
```

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> /bin/ls -aLF
{{ Sortie de la commande "ls" avec les arguments "-aLF" }}
$> echo ${?}
0
```

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> /bin/ls -l -a -F
{{ Sortie de la commande "ls" avec les arguments "alF" }}
$> echo ${?}
0
```

Ne vous contentez pas des tests de la correction, faites vos propres tests en supplément.

✓ Yes

✗ No

Prérequis du 21 sh

Nous allons évaluer les prérequis du 21 sh.

Si l'un au moins de ces tests échoue, alors toute la section est échouée et la correction s'arrête. Effectuez les tests suivants:

- Vérifiez qu'il est possible de se déplacer dans la ligne de commande via divers raccourcis et de l'éditer à l'endroit du curseur.
- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> ls > /tmp/ftsh_ls_out /
$> cat /tmp/ftsh_ls_out
{{ Sortie de la commande "ls" sur la racine du système }}
```

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> < /tmp/ftsh_ls_out cat -e >> /tmp/ftsh_ls_out
$> cat /tmp/ftsh_ls_out
{{ 2 listing de la racine doivent apparaître et le second doit avoir un $ à la fin de chaque ligne }}
```

- Exécutez les commandes suivantes et vérifiez que l'affichage est conforme:

```
$> echo 1 >out >&2 2>err
1
$> echo 2 >out 2>err
$> cat err
$> cat out
2
$>
```

- Exécutez les commandes suivantes et vérifiez que l'affichage est conforme:

```
$> echo non-standard fd > dup_fd
$> cat 4 non-standard fd$
$>
```

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> cat <&4
{{ Message d'erreur indiquant que le descripteur de fichier est invalide }}
$>
```

- Exécutez les commandes suivantes et vérifiez que l'affichage est conforme:

```
$> echo abc >redir_one_to_all
$> cat 9 abc$
$>
```

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> cat <&- abc
$>
```

- Exécutez les commandes suivantes et vérifiez que l'affichage est conforme:

```
$> ls doesnotexist . 2>&1 >/dev/null
ls: doesnotexist: No such file or directory
$> ls doesnotexist . >/dev/null 2>&1
$>
```

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> ls | sort -rn | cat -e
{{ Contenu du dossier courant, trié, avec un '$' a la fin de chaque ligne }}
$>
```

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> base64 < /dev/urandom | head -c 1000 | grep 42 | wc -l | sed -e s/1/Yes/g -e
s/0/No/g
{{ Affiche "Yes" ou "No" aléatoirement }}
$> ps a | grep 'base64' | grep -v 'grep'
```

```
$>
```

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> ls -1; touch test_file; ls -1
{{ Affichage des 2 'ls'. Un fichier supplémentaire, "test_file", doit apparaître
dans la seconde sortie }}
```

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> exit 1 | exit 2 | exit 3; echo "stayin' alive"
stayin' alive
$>
```

Vérifier que le 42sh ne s'est pas terminé et que le prompt est disponible.

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> echo out >&-; echo out2
{{ Message d'erreur facultatif indiquant qu'il est impossible d'écrire sur stdout
}}
out2
$> echo out >&- | echo out2
{{ Message d'erreur facultatif indiquant qu'il est impossible d'écrire sur stdout
}}
out2
$> echo out >&- && echo out2
{{ Message d'erreur facultatif indiquant qu'il est impossible d'écrire sur stdout
}}
$> echo out >&- || echo out2
{{ Message d'erreur facultatif indiquant qu'il est impossible d'écrire sur stdout
}}
out2
```

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> cat << END
heredoc> hello world
heredoc> and good
heredoc> morning!
heredoc> END
hello world
and good
morning!
```

\$>

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> cat << EOF\  
> F  
heredoc> hi  
heredoc> EOF  
hi
```

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> cat > /tmp/heredoc-append << FIN  
heredoc> abc  
heredoc> FIN  
$> cat -e >> /tmp/heredoc-append << FIN  
heredoc> def  
heredoc> ghi  
heredoc> FIN  
$> cat /tmp/heredoc-append  
abc  
def$  
ghi$  
$>
```

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> (cat < heredoc> abd  
heredoc> abc  
heredoc> abb  
heredoc> EOF  
abb$  
abc$  
abd$  
$>
```

Ne vous contentez pas des tests de la correction, faites vos propres tests en supplément. Par exemple, vous pouvez tester des redirections à l'intérieur des pipes.

 Yes

 No

Les built-ins


```
{{ relative/path/of/your/choice }}  
$>
```

- Exécutez les commandes suivantes et vérifiez que l'affichage est conforme:

```
$> cd /tmp  
$> /bin/pwd  
/tmp  
$> cd  
$> /bin/pwd  
/Users/{{login_session}}  
$>
```

- Exécutez les commandes suivantes et vérifiez que l'affichage est conforme:

```
$> cd /tmp  
$> pwd  
/tmp  
$> cd /bin  
$> pwd  
/bin  
$> cd -  
$> pwd  
/tmp  
$>
```

- Exécutez les commandes suivantes et vérifiez que l'affichage est conforme:

```
$> cd -L /tmp; cd -P ..  
$> pwd  
/private  
$>
```

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> type type ls  
{{ Message indiquant que "type" est un builtin et "ls" une commande avec son path }}  
$>
```

Ne vous contentez pas des tests de la correction, faites vos propres tests en supplément. Vous pouvez par exemple tester le bon comportement du built-in "cd" si la variable "CDPATH" est présente dans l'environnement.



Yes



No

Logical operators

Nous allons évaluer l'implémentation des opérateurs logique.

Si l'un au moins de ces tests échoue, alors toute la section est échouée et la correction s'arrête. Effectuez les tests suivants:

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> ls -l && ls
{{ Affichage de la commande "ls" 2 fois avec des paramètres différents }}
$>
```

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> ls doesnotexist || echo 'Notice me senpai'
ls: doesnotexist: No such file or directory
Notice me senpai
$> echo ${?}
0
$>
```

- Exécutez les commandes suivantes et vérifiez que l'affichage est conforme:

```
$> echo 'No error' || echo 'You cant see me'
No error
$> echo ${?}
0
$>
```

- Exécutez les commandes suivantes et vérifiez que l'affichage est conforme:

```
$> false && echo foo || echo bar
bar
$> true || echo foo && echo bar
bar
$>
```

Ne vous contentez pas des tests de la correction, faites vos propres tests en supplément.



Gestion de l'environnement

Nous allons évaluer le support des variables internes et externes, ainsi que l'implémentation des builtins "set", "export" et "unset".

Si l'un au moins de ces tests échoue, alors toute la section est échouée et la correction s'arrête. Effectuez les tests suivants:

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> a=hello b=world; b=42 echo ${a}_${b} && echo ${b}
hello_world
world
$>
```

- Exécutez les commandes suivantes et vérifiez que l'affichage est conforme:

```
$> directory=/ ls_opt=-atr
$> ls ${ls_opt} ${directory}
{{ Sortie de la commande "ls -atr" sur la racine du système }}
$>
```

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> echo ${empty}|cat -e
$
$>
```

- Exécutez les commandes suivantes et vérifiez que l'affichage est conforme:

```
$> set
{{ Ensemble des variables internes au shell et d'environnement }}
$> set | grep -E '(a|b)= '
a=hello
b=world
$>
```

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> env
{{ Ensemble des variables d'environnement uniquement }}
$> env | grep -E '(a|b)= '
```

```
$>
```

- Exécutez les commandes suivantes et vérifiez que l'affichage est conforme:

```
$> export b
$> printenv b
world
$>
```

- Exécutez les commandes suivantes et vérifiez que l'affichage est conforme:

```
$> ONESHOT= env | grep ONESHOT
ONESHOT=
$> env | grep ONESHOT
$>
```

- Exécutez les commandes suivantes et vérifiez que l'affichage est conforme:

```
$> unset a b
$> env | grep -E '(a|b)= '
$> set | grep -E '(a|b)= '
$>
```

- Exécutez les commandes suivantes et vérifiez que l'affichage est conforme:

```
$> unset PATH
$> PATH=/bin:/usr/bin
$> mkdir testdir
$> echo ${?}
0
$> ls -l | grep testdir
testdir
$>
```

- Exécutez les commandes suivantes et vérifiez que l'affichage est conforme:

```
$> true; echo ${?}; false; echo ${?}
0
1
$>
```

Ne vous contentez pas des tests de la correction, faites vos propres tests en supplément. Vous pouvez par exemple tester les options du built-in "export".

✓ Yes

✗ No

Job control

Nous allons évaluer l'implémentation du job control.

Le job control permet de contrôler les processus de manière interactive en permettant de placer des commandes en arrière-plan, de les stopper et de les reprendre en avant-plan.

Si l'un au moins de ces tests échoue, alors toute la section est échouée et la correction s'arrête. Effectuez les tests suivants:

- Exécutez les commandes suivantes et vérifiez que l'affichage est conforme:

```
$> mkfifo fifo
$> ls -lR /usr >fifo 2>&1 &
$> jobs
{{ Message indiquant que la commande "ls" est en cours d'exécution }}
$>
```

- Exécutez les commandes suivantes et vérifiez que l'affichage est conforme:

```
$> emacs -nw &
$> emacs -nw &
$> emacs -nw &
$> emacs -nw &
$> emacs -nw &
$> jobs
{{ Message indiquant que 5 instances d'emacs sont stoppés en arrière-plan }}
$>
```

- Exécutez la commande suivante et vérifiez que l'affichage est conforme:

```
$> fg %{{ un des emacs job number }}
```

Le processus Emacs doit revenir en premier-plan et être fonctionnel. Recommencez autant de fois que nécessaire pour ramener tout les processus "emacs".

- Exécutez les commandes suivantes et vérifiez que l'affichage est conforme:

```
$> jobs
{{ La commande "ls" lancé plus haut doit être la seule restante }}
$> cat -e {{ Sortie de la commande "ls". N'attendez pas la fin, coupez l'affichage avec CTRL-C }}
$> jobs
$>
```

- Exécutez les commandes suivantes et vérifiez que l'affichage est conforme:

```
$> ls -Rl / 2>&1
{{ Affichage de la commande "ls". N'attendez pas la fin, appuyez sur CTRL-Z }}
{{ Message indiquant que la commande est suspendue }}
$> jobs
{{ Message indiquant que la commande est suspendue }}
$>
```

- Exécutez les commandes suivantes et vérifiez que l'affichage est conforme:

```
$> ps a | grep "ls -Rl /" | grep -v 'grep' | cut -d ' ' -f 2
{{ PID de la commande "ls" du test précédent }}
$> kill {{ PID de la commande "ls" du test précédent }}
$> jobs
{{ Message indiquant que la commande s'est terminé }}
$>
```

- Exécutez les commandes suivantes et vérifiez que l'affichage est conforme:

```
$> jobs
$> ps a | grep "ls -Rl /" | grep -v 'grep'
$>
```

Ne vous contentez pas des tests de la correction, faites vos propres tests en supplément. Vous pouvez par exemple tester le built-in "bg".

 Yes

 No

Signaux

Nous allons évaluer la gestion des signaux.

Si l'un au moins de ces tests échoue, alors toute la section est échouée et la correction s'arrête. Effectuez les tests suivants:

- Vérifiez que le shell gère correctement les signaux émis par ses enfants.

Pour cela vous pouvez utiliser la commande suivante:

```
$> python -c 'import os, signal;os.kill(os.getpid(), signal.SIGSEGV)'  
{ { Message indiquant le signal reçu } }  
$>
```

Remplacez "SIGSEGV" par le signal que vous voulez envoyer.

Testez tout les signaux!

Le shell ne doit en aucune façon quitter si un de ses enfants se termine par un signal, même si c'est un "SIGKILL".

- Vérifier que "CTRL-C" dans un prompt vide et avec une commande réaffiche bien un prompt vide.

- Exécutez la commande:

```
$> cat
```

Puis appuyez sur "CTRL-\".

La commande "cat" doit se terminer avec un message qui indique le signal reçu et le prompt être disponible.

Ne vous contentez pas des tests de la correction, faites vos propres tests en supplément.

✓ Yes

✗ No

Partie modulaire

Rappel : vous ne devez évaluer la partie modulaire que si la partie obligatoire est PARFAITE. Il n'y a aucun intérêt à développer des fonctionnalités exotiques sur un shell qui n'assure pas de manière parfaite les fonctionnalités de base ! Les tests pour les sections suivantes se trouvent dans des fichiers séparés. Vous devez effectuer tous les tests. Si l'un au moins de ces tests échoue, alors toute la section est échouée et vous passez à la suivante. Chaque fichier contient un transcript d'une session shell. Vous devez reproduire les commandes listées dans le fichier et vous assurer que l'affichage correspond à ce qui est attendu. Faites attention aux prompts présent dans les fichiers: - "\$> " représente le prompt normal, en attente de commande - "> " représente une commande incomplète qui attend une entrée utilisateur ((heredoc, quote, dquote...))

Inhibiteurs

Dans cette section nous allons évaluer la présence et le bon fonctionnement des inhibiteurs "" (double quote), " (simple quote) et "\" (backslash).

Un prompt "quote>" indique que le shell attend une entrée supplémentaire

pour compléter la commande courante.

Les tests pour cette section se trouvent dans le fichier "42sh.quoting.txt".

Ne vous contentez pas des tests de la correction, faites vos propres tests en supplément.

Vous pouvez par exemple tester ces commandes et vérifier que le shell réagit de la même façon que le shell de référence choisit par le groupe:

```
$> echo foo\  
$> echo "\\ 'abcd\\ '"  
$> echo \'
```

☒ Yes

☐ No

Pattern Matching

Dans cette section nous allons évaluer le bon fonctionnement du globbing ("*", "?", "[", "-", "!").

Regardez dans les sources l'implémentation du globbing, la fonction glob(3) ne doit pas être utilisée.

Les tests pour cette section se trouvent dans le fichier "42sh.pattern_matching.txt".

Faites également vos propres tests!

Vous pouvez par exemple tester comment le shell se comporte si un élément du pattern est échappé (avec "\\") ou si le pattern est entre inhibiteurs ("")

☒ Yes

☐ No

Expansion Supplémentaire

Testez la présence et le bon fonctionnement des expansions du tilde et des paramètres.

Les tests pour cette section se trouvent dans le fichier "42sh.expansions.txt".

Ne vous contentez pas des tests de la correction, faites vos propres tests en supplément. Il existe énormément de tests possibles concernant l'expansion. Soyez inventif !

☒ Yes

☐ No

Commandes groupés et sous-shells

Testez la presence et le bon fonctionnement des sous-shells et commandes groupées:

Les tests pour cette section se trouve dans le fichier "42sh.grouped_commands.txt".

Ne vous contentez pas des tests de la correction, faites vos propres tests en supplément.

Testez également des erreurs de syntaxe, tel que:

```
$> ()  
$> (echo a|)  
$> (; echo b)  
$> (echo c; ())
```

☒ Yes

☒ No

Substitution de commandes

Testez la presence et le bon fonctionnement de la substitution des commandes

Les tests pour cette section se trouve dans le fichier "42sh.command_sub.txt".

Ne vous contentez pas des tests de la correction, faites vos propres tests en supplément. Il existe énormément de tests possibles concernant la substitution de commande.
Soyez inventif !

☒ Yes

☒ No

Expansion Arithmétique

Testez la presence et le bon fonctionnement des expansions arithmétique

Les tests pour cette section se trouve dans le fichier "42sh.exp_arithm.txt".

Testez également que l'expansion s'effectue bien entre double quote ("") et non entre simple quote (''). Testez des valeurs incohérentes, telles que "999999999999999999999999 + 1"

☒ Yes

X No

Substitution de processus

Testez la presence et le bon fonctionnement de la substitution de processus

Les tests pour cette section se trouve dans le fichier "42sh.process_sub.txt".

Ne vous contentez pas des tests de la correction, faites vos propres tests en supplément. Il existe énormément de tests possibles concernant la substitution de processus.

Soyez inventif !

✓ Yes

✗ No

Historique

Testez la presence et le bon fonctionnement de l'historique.

Les tests pour cette section se trouve dans le fichier "42sh.history.txt".

En complément, effectuez les tests suivants:

- Utilisez la commande "fc -l" pour obtenir l'historique des commandes avec leurs index. Puis exécutez une commande avec l'expansion "!". Vérifiez que la bonne commande s'exécute

- Idem qu'au-dessus mais cette fois avec l'expansion "!!". Vérifiez que la bonne commande s'exécute

- Vérifiez que les commandes sont bien enregistrées dans un fichier. Fermez et relancez le shell. Est-ce que l'historique de l'ancienne session est accessible dans le nouveau shell ?

- Exécutez la commande:

```
$> fc -e vim -1 -10
```

Vérifiez que les 10 dernières commandes sont présentes dans vim. Éditez-les si vous le souhaitez, sauvegardez et fermez vim. Les commandes présentes dans vim à la fermeture doivent s'exécuter.

- Vérifiez que la recherche incrémentale, via un le raccourci CTRL-R (ou un autre) fonctionne.

Ne vous contentez pas des tests de la correction, faites vos propres tests

en supplément. Il existe énormément de tests possibles concernant historique et son expansion.

Soyez inventif !

✓ Yes

✗ No

Complétion dynamique contextuelle

Testez la presence et le bon fonctionnement de la complétion dynamique.

Si l'un au moins de ces tests échoue, alors toute la section est échouée et la correction s'arrête. Effectuez les tests suivants:

- Vérifiez que la complétion des commandes dans le "PATH" fonctionne

- Idem pour les builtins

- Vérifiez que la complétion est bien contextuelle. Si vous avez "ls /sbin/" sur la ligne de commande et que vous appuyez sur TAB (ou n'importe quelle autre touche responsable de la complétion) alors seuls les fichiers présent dans le répertoire "/sbin" doivent apparaître.

- Exécutez les commandes:

```
$> abc=def  
$> echo ${a}
```

Vérifiez que la complétion vous propose bien la variable "abc".

- Exécutez les commandes:

```
$> unset a  
$> echo ${a}
```

Vérifiez que la complétion ne vous propose plus la variable "abc".

Ne vous contentez pas des tests de la correction, faites vos propres tests en supplément. Il existe énormément de tests possibles concernant la complétion dynamique.

Soyez inventif !

✓ Yes

✗ No

Mode d'édition Vi/Readline

Testez la presence et le bon fonctionnement des modes d'édition Vi et Readline.

Si l'un au moins de ces tests échoue, alors toute la section est échouée et la correction s'arrête. Effectuez les tests suivants:

- Vérifiez qu'il est possible de changer de mode d'édition avec la commande "set -o vi" ou "set -o readline"
- Vérifiez le bon fonctionnement de tout les raccourcis mentionnés dans le sujet pour les 2 modes.

Ne vous contentez pas des tests de la correction, faites vos propres tests en supplément. Il existe énormément de tests possibles concernant l'édition de ligne. Soyez inventif !

 Yes

 No

Built-ins alias/unalias

Testez la presence et le bon fonctionnement des alias

Les tests pour cette section se trouve dans le fichier "42sh.alias.txt".

Ne vous contentez pas des tests de la correction, faites vos propres tests en supplément. Il existe énormément de tests possibles concernant les alias. Soyez inventif !

Vous pouvez par exemple tester des noms d'alias invalides, comme "=", "-", ou "/". Le shell doit afficher une erreur.

 Yes

 No

Table de hachage

Testez la presence et le bon fonctionnement de la table de hachage et du builtin "hash"

Les tests pour cette section se trouve dans le fichier "42sh.hash.txt".

Ne vous contentez pas des tests de la correction, faites vos propres tests en supplément. Il existe énormément de tests possibles concernant la table de hachage et le builtin "hash". Soyez inventif !

 Yes

 No

Built-in test

Testez la presence et le bon fonctionnement du builtin "test"

Les tests pour cette section se trouve dans le fichier "42sh.test.txt".

En cas d'erreur interne, un code de retour supérieur à 1 est retourné.

Vérifiez le et vérifiez la présence d'un message d'erreur.

Par exemple avec la commande:

```
$> test zzz -eq; echo $?
```

Ne vous contentez pas des tests de la correction, faites vos propres tests en supplément. Il existe énormément de tests possibles concernant la table de hachage et le builtin "test". Soyez inventif !

☒ Yes

☐ No

Bonus

Shell script

Testez la présence et le bon fonctionnement du shell script.

☒ Yes

☐ No

Autocompletion des paramètres

Testez la présence et le bon fonctionnement de l'autocompletion des paramètres

☒ Yes

☐ No

POSIX

L'entièreté du shell doit être POSIX pour valider ce bonus.

Bon courage pour tester la conformité à la norme POSIX :)

☒ Yes

☐ No

D'autres fonctionnalités

Si le 42sh a des fonctionnalités supplémentaires, comptabilisez-les ici. Vous pouvez comptabiliser jusqu'à 5 fonctionnalités bonus. Les bonus doivent être 100%

fonctionnels et ne pas mettre en cause la stabilité du shell.



Rate it from 0 (failed) through 5 (excellent)

Conclusion

Leave a comment on this evaluation

Finish evaluation

General term of use of the site (<https://signin.intra.42.fr/legal/terms/6>)

Privacy policy (<https://signin.intra.42.fr/legal/terms/5>)

Legal notices (<https://signin.intra.42.fr/legal/terms/3>)

Declaration on the use of cookies (<https://signin.intra.42.fr/legal/terms/2>)

Terms of use for video surveillance (<https://signin.intra.42.fr/legal/terms/1>)

Rules of procedure (<https://signin.intra.42.fr/legal/terms/4>)