Protocoles Linux

jeudi 10 avril 2025

13:45

|  |  |  |
| --- | --- | --- |
| NTP | Network Time Protocol > Protocole permettant de distribuer l'heure sur un réseau informatique | <https://doc.ubuntu-fr.org/ntp> |
|  |  |  |

Commandes réseau

mercredi 9 avril 2025

13:39

|  |  |  |  |
| --- | --- | --- | --- |
| Commande | Description | Exemple | Particularité |
| ip addr show | Afficher listing adresse ip sur linux | ip addr show |  |
| ethtool | Affiche et/ou modifie certains paramètres de la carte réseau | ethtool -p eth0 |  |
| fudge | Définir sa propre horloge system | Fudge 127.127.1.0 stratum 10 |  |
| stratum | A indiquer après un fudge pour définir la strate de prise en compte | Fudge 127.127.1.0 stratum 10 |  |
| nmtui | Génère une pseudo interface graphique pour la modification des interfaces réseaux | nmtui |  |
|  |  |  |  |

Commandes Linux

### Lien utile :

<https://www.linuxtricks.fr/pages/bienvenue-sur-linuxtricks>  
<https://readthedocs.com/>  
<https://www.ibm.com/docs/en/aix/7.3?topic=commands>

mardi 15 avril 2025

08:51

|  |  |  |  |
| --- | --- | --- | --- |
| Commande | Description | Exemple | Particularité |
| ./ | Permet d'exécuter un script bash | ./monscript.sh  ~/Documents/MesScripts/monscript.sh | S'assurer d'être dans le dossier correspondant sinon pointer en dur vers le dossier Ne pas oublier de se donner les droits d'exécution |
| ll | Listing des fichiers dans un dossier, plus complet avec les droits sur fichiers et dossiers d'affiché, etc | ll |  |
| ls | Listing des fichiers et dossiers | ls /usr/bin/ ls /usr/bin/ | grep "chains" ls /usr/bin/ > ~/Documents/test.txt ls /usr/bin/ | grep "chains" > ~/Documents/testv2.txt | on peut utiliser | suivi d'un grep pour filtrer via mot clé  on peut utiliser > pour rediriger le flux généré vers un fichier La combinaison de plusieurs paramètres est possible |
| cp  scp | Pour copier des fichiers d'un endroit à un autre | cp text.txt /home/ utilisateur/dossierdedestination/copie\_texte.txt | scp pour les copies en réseau  -r pour copier un dossier |
| mv | Pour couper coller des fichiers d'un endroit à un autre, utilisé aussi pour renommer les fichiers | mv text.txt /home/ utilisateur/dossierdedestination/copie\_texte.txt | besoin d'être dans le repertoire courant pour renommer le fichier |
| rm | Pour supprimer des fichiers et dossiers | rm -r ~/Documents/test/\*  rm `ls` | -r pour le récursivité, donc le dossiers et tous ses sous dossiers et contenus associés  -f pour forcer la suppression sans requête  -i pour demander une confirmation supplémentaire avant chaque suppression  -v pour d'indiquer l'action en cours  `` pour supprimer la sortie de la commande contenue entre ``, voir exemple |
| man | Consulter l'aide d'une commande | man lsblk |  |
| sudo su - | Connexion en utilisateur élevé (admin) | sudo su - |  |
| su - | Connexion en root | su - |  |
| ./ | Execute un script dans le dossier en cours | ./make.sh |  |
| vi vim | Edite un fichier, puissant | vi/link\_install | i pour insertion  # pour commenter  echap pour quitter le mode insertion  wq pour sauvegarder en quittant : pour entrer dans "l'invite de commande" de vi q! pour quitter sans sauvegarder |
| ~ | Raccourci pour $home | ~/Git/… |  |
| dnf | Installation de logiciels et de paquets dans la distribution, plus performant que rpm | dnf install paquet    dnf provides /usr/bin/fichier    dnf search NetworkManager    dnf repolist    dnf clean all | install pour installer un paquet  remove pour désinstaller un paquet provides pour rechercher un paquet contenant un fichier  upgrade pour mettre à jour un paquet  dnf upgrade pour mettre à jour la distribution  --exclude= paquet update pour exclure un paquet de la mise à jour  search pour faire une recherche via mot clé  repolist pour afficher les repo enabled  repolist --all pour afficher tous les repo  config-manager pour configurer les repos présents, enable et disable  clean all pour vider le cache dnf |
| lsblk | Faire un listing des devices et des points de montages (usb, disques dur, etc) | lsblk | -f pour la liste complète  Se référer à la ligne mountpoint pour voir vers ou pointer pour les commandes suivantes sur le device ( par exemple /run/media/username est le chemin de base) |
| ip | Permet la configuration de tout ce qui touche à l'ip | ip addr sh ip route | addr sh pour afficher les adresses route pour afficher les routes |
| dd | Permet de créer une support bootable à partir d'un iso | sudo dd if=home/aicardic/iso.iso of=/dev/sda status=progress | if = input file, donc l'iso à implémenter of = output file, donc le support à flasher status = affiche la progression du flash |
| add | Permet l'ajout d'éléments selon les arguments présents | add route add user |  |
| checkisomd5 | Pour vérifier l'intégrité d'un iso | checkisomd5/dev/sda |  |
| chown | Commande pour changer le propriétaire du dossier/fichier | sudo chown aicardic:aicardic /run/media/aicardic/SSK-EXT4/ | aicardic:aicardic, la valeur avant le deux point pour le login name et après pour le user group |
| chmod | Commande pour changer les permissions sur un dossier/fichiers | chmod ugo+rw exemple.txt | u pour user g pour group o pour others a pour all  r pour droit de lecture w pour droit d'écriture x pour droit d'exécution |
| grep | Commande pour chercher des motifs et des chaines de caractères | grep "test" exemple.txt |  |
| tee | Commande pour rediriger la sortie d'une commande vers le terminal et un ou plusieurs fichiers | lsblk -f | tee ~/Downloads/lslbk.txt  ls -l /| tee racine.txt | grep lrwx  ping yum.home.arpa | tee -i ping-gateway.txt | On peut enchainer une autre commande derrière  -a pour ne pas écraser le ou les fichiers cibles mais écrire à la suite  -i pour ignorer les interruptions |
| echo | Permet de répéter ce qu'on lui indique | echo $?  echo "var1"  echo "Bonjour" echo $((a=5+5 , b=10+5 , 21+21)) echo a=$a b=$b | $? pour afficher le code de retour de la commande précédente, ne pas confondre avec ce qui est censé être affiché  echo $(()) pour faire du calcul  echo $[[]] pour faire du test de variables  echo ${!nom-tableau[@]} pour les tableaux |
| printf | Plus complet qu'echo, à privilégier | printf "%s\n" "${tn[@]}" | "%s\n" pour préciser que c'est "all strings" et le \n pour le retour chariot, à mettre avant chaque contenu entre "" \t pour effectuer des tabulations |
| service syste.. warning::  Ne pas oublier la commande  .. code-block::  #!/bin/bash  En début de script, sans quoi ce dernier ne se lancera pas. Ne pas confondre avec les commentaires marqués \*#\*.mctl.. warning::  Ne pas oublier la commande  .. code-block::  #!/bin/bash  En début de script, sans quoi ce dernier ne se lancera pas. Ne pas confondre avec les commentaires marqués \*#\*. | Commande pour modifier/paramétrer/utiliser les systèmes de la distribution (par exemple) | service network restart systemctl restart network-etc | enable pour activer disable pour désactiver  restart pour redémarrer  --now pour exécuter la commande de suite |
| nmtui | Génère une pseudo interface graphique pour la modification des interfaces réseaux | nmtui |  |
| rpm | Commande pour gérer les paquets rpm | rpm [option] [Nom du paquet] rpm -qi Paket1.rpm | Principales options :  -e pour effacer un paquet -i pour installer un paquet -K pour contrôler l'intégrité d'un paquet -q pour afficher toutes les informations disponibles à propos des paquets installés.  -U pour actualiser un paquet rpm -V pour vérifier l'intégrité d'un paquet et inclure les fichiers erronés |
| cat | Commande qui permet d'afficher, de gérer de combiner et de manipuler des fichiers texte. | cat [OPTION] [NOM DU FICHIER] cat /etc/passwd cat /etc/group  cat >test1/essai.txt <<OEF text1 text2  text3  text4 EOF | Le dernier exemple montre bien que l'on peut écrire dans un fichier via cat, pensez aux chevrons et a spécifier le début et la fin du fichier en lui-même avec EOF  On peut lire plusieurs fichiers d'un coup, ainsi :  cat ~/test1/essai.txt ~/test2/adresse.c |
| less | Permet de consulter un fichier tout en gardant le focus dessus | less fichier.sh |  |
| lshw | Permet de sortir une liste de la configuration physique de la machine | lshw -html > ~/downloads/test.html | -html pour écrire dans un fichier HTML  > pour pointer vers le fichier en question |
| ssh | Permet de se connecter en SSH | ssh dir.home.arpa |  |
| useradd | Permet de créer un utilisateur | useradd -c "Clement Aicardi" -G navalgrp,testgrp aicardic | -c en commentary pour spécifier le nom qui s'affiche sur la page de login -G pour ajouter à des groupes existants |
| usermod | Permet de modifier un utilisateur existant | usermod -aG cslab,wireshark naval | -aG pour ajouter à des groupes  écrire les groupes à rejoindre ensuite, les séparer par des virgules, et finir après l'espace par l'utilisateur concerné |
| passwd | Permet de modifier ou de créer un mot de passe | echo aicardicpwd | passwd --stdin aicardic | echo pour l'afficher, le pipe pour l'implémenter dans le passwd  --stdin pour  -e pour le rendre expiré, pratique pour les premières connexions  -d pour supprimer un mot de passe  -a pour le modifier dans tous les module |
| sha256sum | Permet de vérifier l'intégrité d'un fichier avec le sha | sha256sum -c iso.iso.sha256 | c pour un check complet On doit se trouver dans le dossier en question |
| ldapadd | Permet d'ajouter un utilisateur au ldap |  |  |
| ldapmodify | Permet de modifier les entrées ldap |  |  |
| ldapsearch | Permet de chercher dans le ldap |  |  |
| cd people | Permet d'afficher et de naviguer entre les utilisateurs |  |  |
| less | Permet de consulter un fichier texte | less ~/.local/bin/installCSLab.sh |  |
| awk | Permet d'appliquer un certain nombre d'actions sur un fichier | awk 'END {print NR}' fichier | <https://www.shellunix.com/awk.html> |
| git | Permet de gérer les dépôts git | git -C ~/Git/genoraclelinux9iso/ pull --force  git clone [https://depot.git](https://depot.git/) ~/Git/genOracleLinux9Iso    git init  git add .gitignore  git commit -m "message\_commit"    git remote add origin <https://github.com/username/exempleproject>  git push -u origin main/master | -C pour préciser le path --force pour forcer la modification  clone pour cloner un dépôt via url dans un dossier pull pour télécharger depuis un autre dépôt dans un dépôt ou une branche locale clone pour télécharger l'entièreté du dépôt  git init pour configurer le dossier en cours comme dépôt git  git status pour afficher le statut d'un dépôt git    git add pour ajouter des dossiers et fichiers à un commit avant de l'envoyer sur le dépôt distant    add . pour ajouter tous les changements dans le répertoire actuel    add -A pour ajouter tout le dépôt    git commit pour enregistrer les modifications dans le dépôt, -m pour spécifier un message  git remote add origin <URL> pour faire le lien entre les dépôts git local et distant    git push pour mettre à jour le dépôt distant avec les modifications du local remontées dans un commit  -u configure la branche locale pour qu'elle suive automatiquement la branche distante    origin = nom du dépôt distant, convention de nom, ce n'est pas une commande    à faire une fois, ensuite push et pull suffisent |
| .. warning::  Ne pas oublier la commande  .. code-block::  #!/bin/bash  En début de script, sans quoi ce dernier ne se lancera pas. Ne pas confondre avec les commentaires marqués \*#\*.clear | Permet de nettoyer la console pour y voir plus clair | clear |  |
| tc |  |  |  |
| nmcli |  |  |  |
| mkdir | Permet de créer un dossier | mkdir /srv/dossier  mkdir -p ~/Documents/test/{test1,test2} mkdir -p ~/Documents/test/test{1..4} | -p pour créer des sous dossiers même si le dossier parent n'existe pas  entre accolades {} pour créer plusieurs dossiers d'un coup, séparés par une virgule  {1..x} pour créer un nombre de dossiers compris entre 1 et x inclus nommés de manière logique |
| touch | Permet de créer des fichiers vides et de modifier le timestamp | touch test.txt  touch test{1..4}.txt  touch -am test.txt  touch -t 202003081047.30 test.txt  touch -r '8 Mar' test.txt | {1..x} pour créer un nombre de fichier compris entre 1 et x inclus nommés de manière logique  -a pour modifier l'heure d'accès  -m pour modifier l'heure de modification  -c pour modifier le temps d'accès sans créer de nouveaux dossier  -t pour définir temps d'accès et de modification spécifique (format CCYYMMDDhhmm.ss)  -r pour utiliser un autre fichier en timestamp de référence  -d pour spécifier la date et l'heure format string  Pour créer un fichier en code dur sans se rendre dans le repertoire en cd, faire comme ceci : touch ~/test1/adresse.txt |
| grubby | Permet de lire et modifier le fichier grub.cfg Permet de modifier les entrées du menu grub, etc | grubby --default-kernel | <https://tinyurl.com/yc6f6v84> |
| env | Permet de lister les variables | env |  |
| nom\_de\_variable + = + "var\_content" pour créer une variable | Permet de créer une variable | variable="test" variable="test2" | Ne jamais oublier les guillemets  Pour modifier la valeur d'une variable, lui affecter une nouvelle valeur |
| "$nom\_de\_variable" | Permet d'appeler une variable | echo "$nom\_de\_variable" | toujours utiliser les guillemets pour une syntaxe propre |
| unset | Permet d'effacer la mémoire d'une variable spécifique | var1="Bonjour" unset var1 | A manipuler avec précaution |
| export | Permet d'exporter la variable à l'environnement des programmes qui seront lancés par le shell | export NOM\_DE\_VARIABLE export NOM\_DE\_VARIABLE="var\_content" test\_var = "Hello World" | Possible de créer et exporter la variable directement, voir exemple |
| Readonly | Protège la variable concernée pour la figer. Lui évite d'être réaffectée par une nouvelle valeur | var1=toto  var2=  echo "$var1 $var2"  readonly var1 var2  var1=titi  var2=titi |  |
| & | Permet de faire tourner un programme en arrière-plan sans bloquer le shell | iceweasel & | Ajouter ! pour éviter que le programme s'arrête de tourner même quand le shell est fermé  diswon prg\_name pour "revenir en arrière" et ajouter le ! |
| && .. code-block:: bash  #!/usr/bin/env python    -m pour spécifier qu'on souhaite utiliser un module de Python et non un fichier script. | Code retour permettant d'exécuter le second programme si le premier s'est exécuté correctement | prg\_1 && prg\_2 |  |
| || | Code retour permettant de d'exécuter le second programme si le second ne s'est pas exécuté correctement | prg\_1 || prg\_2 |  |
| echo $SHELL | Permet d'afficher à l'écran le shell utilisé | echo $SHELL | Fonctionne aussi comme ceci : env | grep SHELL |
| "" ou '' | Permet de décider ou non si les variables contenus dans la chaine de ligne de commande sont interprétées ou non  " " oui, ' ' non | variable = 'secret' echo "mon mot de passe est $variable" retour = mon mot de passe est secret echo 'mon mot de passe est $variable' retour = mon mot de passe est $variable |  |
| `` | Permet de délimiter une commande à exécuter Noms de variable et commandes sont donc exécutées | echo `variable="connu"; echo "mon mot de passe est $variable."` | ; pour séparer les éléments un à un |
| ${} | Permet de délimiter une variable dans ou d'une chaine de caractères | var="C'est le" echo "${var} reste de la chaine de caractères."  retour prompt : C'est de le reste de la chaine de caractères. | La bonne syntaxe inclus toujours accolades et guillemets  Le # juste après l'accolade permet de connaitre la longueur de la chaines stockées |
| ; | Permet de séparer les instructions | test -z "mot" ; echo $? | Vérifier convenablement les espaces ou non |
| bash --version | Permet de connaitre la version de bash | bash --version |  |
| whereis nom\_commande which nom\_commande | Permet de d'afficher le chemin d'une commande ainsi que celui de sa page man dans le cas de whereis | whereis lsblk  which grep |  |
| type | Permet de connaitre le statut d'une commande, si elle est interne ou externe | type cd | /bin/commande signifie que c'est une commande externe |
| case esac if fi  for done select done until done while done {...}  (...)  ((...))  [...]  [[...]] | Commande composées permettant de mettre en place boucles, conditions etc | fi condition; then commande1; else commande2; fi |  |
| pstree | Permet d'observer la hiérarchie des processus | pstree -p | -p pour afficher le PID en plus de la hiérarchie |
| top | Permet d'observer la liste des processus de manière dynamique | top |  |
| ps | Permet de présenter un cliché instantané des processus en cours | ps ax  ps -A | grep codium | ax pour lister les processus actifs  axu pour lister les processus actifs avec leurs utilisateurs  axum pour lister les processus actifs avec les infos sur les utilisateurs et sur l'occupation mémoire  -A pour sélectionner tous les processus |
| kill | Permet de tuer un processus avec son PID | kill 8768 | On peut trouver les PID avec l'argument -A et grep le processus concerné |
| EOF | Permet de définir des documents en ligne, c’est-à-dire saisir plusieurs lignes simultanément sur l'entrée standard et tant que le mot OEF n'est pas lu, la commande continue. | cat <<EOF >fichier  texte  …  texte  EOF | << pour spécifier la commande visée par EOF |
| declare -a | Permet de créer un tableau | declare -a nom-tableau=(valeur0 valeur1 valeur2 …) | autre syntaxe :  nom-tableau(valeur1, valeur2, …) nom-tableau=([indice0]=valeur0 [indice1]=valeur1 …) |
| read | Permet de lire les entrées standard et affecte les valeurs saisies dans la ou les variables passées en argument Permet de demander à l'utilisateur d'utiliser l'entrée standard  **Il faut bien comprendre que le read permet de créer des variables différemment**  **On peut déclarer des variables successivement et on peut utiliser ces variables avant de les déclarer.** | read var1 var2 a bientôt echo $var1 a echo $var2 bientôt    printf "%s\n" "Bonjour, qui êtes-vous ?" read nom printf "%s\n" "Enchanté, ${nom} !"  read -p "Entrez votre nom et prénom : " nom prenom printf "%s\n" "Bonjour ${prenom} ${nom}." | -a pour indexer tous les noms dans des arrays  Pratique pour que l'utilisateur se serve de l'input standard -p pour insérer un message avant l'attente de la valeur que l'utilisateur doit entrer -n pour limiter le nombre de caractères  -t pour limiter le temps autorisé pour la saisie  -s pour ne pas afficher le texte saisi  On peut utiliser plusieurs options |
| tail | suivre l'évolution d'une commande, d'un fichier | tail -f ~/bin/data/xxx.sh | -f pour follow |
| set | La commande set permet d'affecter une valeur provisoire à un ou plusieurs paramètres de position | set un deux trois  echo "${1}" "${2}" "${3}"  echo "${#}"  echo "il y a deux paramètres de position : "${1}" ; "${2}"."  echo " et les enregistre dans une variable prédéfinie pour le shell courant." | set -- pour retirer les paramètres |
| exit | Permet de remplacer le code de retour de la dernière commande d'un script | var=bonjour if [ ${var} == bonjour ] ; then  printf "%s\n" "${var} est correspond à bonjour" exit 1 fi |  |
| pip install <nom\_package> | Permet de télécharger et installer des packages Python | pip install pandas | On peut installer plusieurs packages d'un coup, les séparer par des espaces -r pour spécifier un requirements.txt pour installer tous les packages listés |
| pip freeze | Permet d'afficher la liste des packages Python installés | pip freeze |  |
| source | Permet de sourcer un environnement de travail, notamment les environs virtuels comme ceux crées en Python | source env/bin/activate | env en exemple est le nom de l'environnement virtuel crée |
| deactivate | Permet de quitter l'environnement virtuel en cours | deactivate |  |
| hostnamectl | Permet d'afficher les infos de la machine, notamment son nom | hostnamectl | set-hostname <nom\_machine> pour modifier en dur le nom de la machine **Penser à impérativement modifier le fichier /etc/hosts** |
| dig | Installé avec sudo dnf install bind-utils, permet de d'interroger les serveurs DNS | dig +short z420.home.arpa | +short pour afficher uniquement les informations essentielles |
| mapfile | Permet de lire les lignes d'entrée dans une variable "array", tableau | mapfile MONFICHIER < exemple.txt  echo ${MONFICHIER[@]} echo ${MONFICHIER[0]}    mapfile GEEKSFORGEEKS < <(printf "Item 1\nItem 2\nItem 3\n") echo ${GEEKSFORGEEKS[@]}  sortie : Item 1 Item 2 Item 3 | < <() pour capturer l'ouptut dans le tableau en question  -t permet d'éviter d'intégrer le délimiteur (généralement le retour chariot) dans la variable  -n Pour lire le nombre de lignes spécifiées, si le compteur est à zéro, toutes les lignes sont lues |
| sed [OPTIONS] 'SCRIPT' [FICHIER…] | Commande qui permet d'éditer un texte en non interactif, peut éditer des fichiers ou des flux d'entrées selon des commandes spécifiées. | exemple.txt :  =============    Bonjour tout le monde.  Cette ligne contient une erreur.  Une autre ligne.  Encore une autre ligne.  Fin du fichier.    ---------------------    sed "s/erreur/correction/" exemple.txt    sed "s/ligne/phrase/g" exemple.txt    sed "/monde/i Ligne ajoutée avant." exemple.txt    sed "/fichier/a Ligne ajoutée après." exemple.txt    sed "/monde/c C'est une ligne complètement nouvelle." exemple.txt | Options :    -i ou --in-place pour modifier les fichiers directement, sans sauvegarde par contre  -e pour spécifier plusieurs commandes  -n pour désactiver la sortie automatique  -f pour charger un fichier contenant des commandes sed      Commandes principales dans un script :    s pour remplacer un texte par une autre dans une ligne  g pour remplacer toutes les occurrences  d pour supprimer les lignes correspondantes selon un modèle  a pour ajouter une ligne après la ligne correspondante  i pour insérer une ligne avant la ligne correspondante  c pour remplacer entièrement une ligne correspondant à un motif  q pour quitter après un certain nombre de lignes ou un motif |
| getopts optstring variable | Commande permettant d'analyser des options dans un script shell. Utile pour traiter des arguments passés au script sous forme d'options (-a -b etc)    otpstring définit les options possibles. Si une option doit accepter une valeur, elle est suivie d'un ":"  variable correspond au nom de la variable dans laquelle l'option sera stockée | while getopts ":ab:" opt; do  case $opt in  a)  echo "Option -a activée"  ;;  b)  echo "Option -b activée avec l'argument : $OPTARG"  ;;  \?)  echo "Option invalide : -$OPTARG" >&2  ;;  esac  done | : au début pour gérer les erreurs soi-même  a est une option sans argument  b: est une option qui nécessite un argument (stipulé par ":")  $OPTARG équivaut à la valeur associé à l'option -b |
|  |  |  |  |

Métacaractères et caractères spéciaux d'échappement linux

jeudi 24 avril 2025

15:31

|  |  |  |  |
| --- | --- | --- | --- |
| Métacaractère | Description | Exemple | Particularité |
| \* | Signifie "n'importe quelle chaîne de n'importe quels caractères" | for f in \*; do echo "$f"; done | les fichiers et dossiers commençant par un "." ne sont pas affectés. |
| ? | Signifie "un caractère quelconque et un seul" | ls ????? | affichera le nombre de caractères remplacé par ? y compris les extensions de fichiers |
| \ | Permet de d'échapper les caractères spéciaux, d'échappement et le double guillemet "" Pratique pour échapper les variables notamment | \$var3 ($var3) correspond à \$var4 ($var4). |  |

Expressions régulières Linux

lundi 5 mai 2025

11:18

|  |  |  |  |
| --- | --- | --- | --- |
| Expression | Description | Exemple | Particularité |
| vi vim | Edite un fichier, puissant | vi/link\_install | i pour insertion  # pour commenter  echap pour quitter le mode insertion  wq pour sauvegarder en quittant : pour entrer dans "l'invite de commande" de vi q! pour quitter sans sauvegarder |
| grep | Permet d'explorer un ou une série de fichiers d'un ou plusieurs répertoires à la recherche de textes filtrés par une expression régulière donnée Peut être étendu à tous les fichiers d'un répertoire ou encore à tous les fichiers correspondant à un "masque" | ll | grep "test.txt"  grep "test" exemple.txt  grep c ~/test1/essai.txt  grep a ~/te\*/\* | Fonctionne pour les caractères, lettres, strings, integer, etc  -n ajoute le numéro de ligne  -i pour ignorer la case    -v pour la négation, c’est-à-dire tout afficher sauf ce qui contient la chaine de caractères donnée    -c pour donner le nombre de lignes où la chaîne apparait sans afficher les lignes  pour regexp, c'est ici : [Image sur la page « Raccourcis clavier/Divers »](onenote:#Raccourcis%20clavier\Divers&section-id={8DFC3A21-C5E9-48F7-B65E-A053DEFE0A5F}&page-id={3232CB37-6B45-4E59-BA31-FDFE1A7FDF53}&object-id={65961C9B-4366-4EA1-AE5C-9EE87F6E2846}&11&base-path=C:\Users\aicardic\Documents\Blocs-notes%20OneNote\Mon%20bloc-notes\Linux.one) |
| expr |  |  |  |
| sed | Permet de lire les lignes d'une fichier une à une sans traiter nécessairement tout le texte du fichier en cours, appliques des commandes d'édition et renvoie les lignes résultantes sur la sortie standard **sans modification du fichier traité.** |  |  |
| awk |  |  |  |

Instructions bash

vendredi 25 avril 2025

10:57

|  |  |  |  |
| --- | --- | --- | --- |
| Condition | Description | Exemple | Particularité |
| if condition;  then  commande1; else commande2; fi | Signifie que le code ne sera exécuté uniquement si la condition testée est vraie | if "$var1" = "$var2"  then  echo "ça fonctionne" else echo "ça ne fonctionne pas"  fi  if [ -e /bin/echo ] && [ -x /bin/echo ];  then  /bin/echo "Bonjour le monde."  else  echo "Bonjour le monde."  fi | Peut être traduit par si [condition],  ensuite sinon  fin  Les conditions peuvent être très variées (chemin d'accès, variables, etc)  Les conditions peuvent être multiples, séparées par un &&   [ ] pour tester des strings, comme par exemple des variables ou des chemins d'accès, ne pas oublier les espaces entre les conditions et les crochets Chemin d'accès avec paramètres, -e pour savoir si le chemin existe, -x pour savoir si l'exécutable est bien exécutable |
| while condition; do commande; done | Semblable à if, sans possibilité de else cependant | while [ "${USER}" = "aicardic" ] do printf "%s\n" "Bonjour !" break done | utiliser break pour éviter une boucle infinie |
| for <element> in <listedElements> do commande à exécuter pour chaque éléments done | Sert à automatiser les tâches répétitives, effectue les opérations plusieurs fois | for i in "les fleurs" "le lapin" "le chou" "la salade" "le staff NG"; do printf "%s\n" "J'aime ${i}." done | i (nom pris au hasard) correspond aux éléments de la liste, on peut préciser les arguments avant de lancer le script juste après le nom de celui-ci |
| case $variable in modele1)  commande1 ;; modele2) commande2 ;; modele3|modele4|modele5) commande3 ;; esac | Permet d'orienter la suite du programme en fonction d'un choix de différentes valeurs Recommandé quand il y a un nombre important de choix vis-à-vis de if | Read -r x ; case "${x}" in bonjour) printf "%s\n" "Bonjour ${USER}" ;; aurevoir) printf "%s\n" "Au revoir ${USER}" ;; \*) printf "%s\n" "Que voulez-vous dire par ${x} ${USER} ?" esac | Une fois le programme lancé, le prompt invite l'utilisateur à entrer une valeur Le \*) signifie "tout ce qui n'est pas" vis-à-vis des modèles du dessus |
|  |  |  |  |

La commande **$[…]** équivalente à la commande **$((..)) *ne doit plus être utilisée***.

* + Sans précision les nombres sont en base 10 (décimal).
  + Préfixé d'un **0**, le nombre est un octal (base 8).
  + Préfixé de **0x** (ou **0X**), le nombre est un hexadécimal (base 16).

Si non, la syntaxe suivante peut-être utilisée :

* + [**base#**]**n**

Avec :

* + **base#** où **base** est un nombre en décimal désignant la base utilisée (de 2 à 64).
  + **n** Le nombre lui même :
    - Jusqu'à la base 10, le nombre contient que des chiffres.
    - De la base 11 à la base 36, les lettres minuscules ou majuscules peuvent être utilisées indifféremment.
    - De la base 37 à la base 64, les lettres minuscules, majuscules, @ et \_ sont à utiliser dans cet ordre.

Opérateurs bash

vendredi 25 avril 2025

15:47

**Avec bash c'est l'inverse, faux est une valeur non nulle et vrai correspond à 0.**

|  |  |  |  |
| --- | --- | --- | --- |
| Opérateur | Description | Exemple | Particularité |
| --- | Opérateur arithmétique | --- | --- |
| +expr | Opérateur pour donner la valeur de expr | +24 | Signe positif avant un nombre |
| -expr | Opérateur pour donner l'opposé de expr | -24 | Signe négatif avant un nombre |
| expr1+expr2 | Opérateur d'addition | 1+2 |  |
| expr1-expr2 | Opérateur de soustraction | 2-1 |  |
| expr1\*expr2 | Opérateur de multiplication | 2\*1 |  |
| expr1/expr2 | Opérateur de division | 24/2 |  |
| expr1%expr2 | Opérateur de modulo | 24%5 | Reste de la division de expr1 par expr2 |
| expr1\*\*expr2 | Opérateur arithmétique de puissance | 2\*\*2 | expr1 multiplié par lui -même expr2 fois |
| --- | Opérateur d'affectation | --- | --- |
| ++var | Affectation par la pré-incrémentation à 1 | var=var+1 puis retourne var echo $((++var)) |  |
| var++ | Affectation par la post-incrémentation à 1 | retourne var, puis var=var+1  echo $((var++) |  |
| --var | Affectation par la pré-décrémentation à 1 | var = var-1 puis retourne var  echo $((--var)) |  |
| var-- | Affectation par la post-décrémentation à 1 | retourne var, puis var=var-1 echo $((var--)) |  |
| var=expr | Simple affectation | var=1 | Affectation de variable, voir ici :  [nom\_de\_variable + = + "var\_content" pour](onenote:#Commandes%20Linux&section-id={8DFC3A21-C5E9-48F7-B65E-A053DEFE0A5F}&page-id={B66C1324-6EA1-4635-A33A-537E08939C6A}&object-id={B689B2D8-D0BA-44AD-9D44-D7F49FB0F8BD}&6B&base-path=C:\Users\aicardic\Documents\Blocs-notes%20OneNote\Mon%20bloc-notes\Linux.one) |
| var+=expr | Affectation par l'incrémentation | var=var+expr  echo $((var+=11)) |  |
| var-=expr | Affectation par la décrémentation | var=var-expr echo $((var-=11)) |  |
| var\*=expr | Affectation par la multiplication | var=var\*expr echo $((var\*=11)) |  |
| var/=expr | Affectation par la division | var=var/expr  echo $((var/=11)) |  |
| var%=expr | Affectation par modulo | var=var%expr echo $((var%=11)) |  |
| var<<=expr | Réaffectation par le décalage bit à bit à gauche | var = var << expr echo $((var<<=11)) |  |
| var>>=expr | Réaffectation par le décalage bit à bit à droite | var = var >> expr echo $((var>>=11)) |  |
| var&=expr | Affectation par le ET binaire | var = var & expr echo $((var&=11)) |  |
| var|=expr | Affectation par le OU binaire | var = var | expr echo $((var|=11)) |  |
| var^=expr | Affectation par le OU exclusif binaire | var = var ^ expr echo |  |
| , | Opérateur virgule pour séparer les expressions afin de les réaliser à la suite | Echo $((45+5 , 55+12 , 21+21)) |  |
| --- | Opérateurs logiques | --- | --- |
| !(expr) | Réalise des opérations sur les booléens. Négation logique, retourne l'inverse de expr | echo "!(0) = $((!(0)))" | !(1) = 0(NON vrai = faux) !(0) = 1(NON faux = vrai) |
| expr1 && expr2 | Réalise des opérations sur les booléens. ET logique | echo $((1 && 1)) | 1 && 1 = 1 (vrai ET vrai = vrai)  1 && 0 = 0 (vrai ET faux = faux)  0 && 1 = 0 (faux ET vrai = faux)  1 && 1 = 0 (faux ET faux = faux) |
| expr1 || expr2 | Réalise des opérations sur les booléens. OU logique | a=0  echo $((a && (a=10) || (a=20) )) a=$a | 1 && 1 = 1 (vrai OU vrai = vrai)  1 && 0 = 1 (vrai OU faux = vrai)  0 && 1 = 1 (faux OU vrai = vrai)  0 && 0 = 0 (faux OU faux = faux) |
| --- | Opérateurs relationnels |  |  |
| == | Vrai si égale | (( 45 == 45 )) echo $? |  |
| != | Vrai si différent | (( 45 != 15 )) echo $? |  |
| > | Vrai si plus grand que | (( 51 > 39 )) echo $? |  |
| >= | Vrai si plus grand ou égal que | (( 51 >= 39 )) echo $? |  |
| < | Vrai si plus petit que | (( 21 < 93 )) echo $? |  |
| <= | Vrai si plus petit ou égal que | (( 21 <= 93 )) echo $? |  |
| --- | Opérateurs bit-à-bit | --- | --- |
| ~ expr | Les opérateurs bit-bit opèrent sur des nombres binaires. Les nombres sont convertis en binaire, l'opération est appliqué puis le résultat est reconverti en décimal  Négation binaire | echo $((~42)) | Renvoie le complément à 1 de expr (~1 -> 0 ; ~0 -> 1) |
| expr1 & expr2 | ET binaire | echo $((41&56)) | Renvoie le ET bit-à-bit entre expr1 et expr2 (1 ET 1 -> 1 ; 0 ET 0 -> 0 ; 1 ET 0 -> 0 ; 0 ET 1 -> 0) |
| expr1 | expr2 | OU binaire | echo $((41|23)) | Renvoie le OU bit-à-bit entre expr1 et expr2 (1 OU 1 → 1 ; 0 OU 0 → 0 ; 1 OU 0 → 1 ; 0 OU 1 → 1) |
| expr1 ^ expr2 | OU binaire exclusif | echo $((41^23)) | Renvoie le OU bit-à-bit exclusif entre expr1 et expr2  (1 XOR 1 → 0 ; 0 XOR 0 → 0 ; 1 XOR 0 → 1 ; 0 XOR 1 → 1) |
| expr1 << expr2 | Décalage binaire à gauche | echo $((41<<23)) | Renvoie le décalage de expr2 bit(s) à gauche de expr1  (expr2 0 ajoutés(s) à droite d'expr1) |
| expr1 >> expr2 | Décalage binaire à droite | echo $((41>>1)) | Renvoie le décalage de expr2 bit(s) à droite de expr1  (expr2 bit(s) tronqué(s) à droite d'expr1) |
| --- | Opérateurs conditionnels | --- | --- |
| expr1 ? expr2 : expr3 | Permettent de réaliser des opérations sous conditions. Les opérations sélectionnées sont déterminées selon une structure **si … alors … sinon**… mais en plus succin. | echo $((24<42 ? 5 : 10)) |  |
| --- | Opérateurs de contrôle | --- | --- |
| & | Toutes les commandes sont exécutées parallèlement |  |  |
| && | Opérateur de succession conditionnée, la commande suivante est exécutée uniquement si la précédente a fonctionné | ll && grep "test" |  |
| ; | Opérateur de succession non conditionnée, chaque commande est exécuté l'une après l'autre même si l'une d'elle a mal fonctionné | ll ; grep "test" |  |

Tableaux Bash

lundi 28 avril 2025

12:34

|  |  |  |  |
| --- | --- | --- | --- |
| Commande | Description | Exemple | Particularité |
| declare -a declare -ai | Permet de créer un tableau | declare -a nom-tableau=(valeur0 valeur1 valeur2 …) | autre syntaxe :  nom-tableau(valeur1, valeur2, …) nom-tableau=([indice0]=valeur0 [indice1]=valeur1 …) -ai pour créer un tableau d'entiers |
| read -a | Permet de lire les entrées standard et affecte les valeurs saisies dans la ou les variables passées en argument | read var1 var2 a bientôt echo $var1 a echo $var2 bientôt |  |
| --- | Récupération de valeur(s) | --- | --- |
| ${nom-tableau[indice]} | Valeur d'une case | printf "%s\n" "${tab[1]}" | Toujours mettre les variables entre parenthèses |
| ${nom-tableau[\*]} ${nom-tableau[@]} | Tous les éléments du tableau sont accessibles avec chacune de ces deux syntaxes | printf "%s\n" "${tab[@]}" printf "%s\n" "${tab[\*]}" |  |
| ${#nom-tableau[\*]} ${#nom-tableau[@]} | Le nombre d'éléments d'un tableau set accessible par chacune de ces deux syntaxes | printf "%s\n" "${#tab[@]}" printf "%s\n" "${#tab[\*]}" |  |
| ${!nom-tableau[\*]} | Pour récupérer le numéros des indices qui ont été utilisés dans un tableau | printf "%s\n" "${!tab[@]}" |  |
| nom-tableau3=("${nom-tableau1[@]}" "${nom-tableau2", ...) | Permet de concaténer dans un seul tableau deux ou plusieurs tableaux | array=("${tab1[@]}" "${tab2[@]}") |  |
| tableau[${#tableau[@]}]=element | Permet d'ajouter un élément en fin de tableau | tab[${#tab1[@\*}]=nom4  Autre syntaxe :  tab+=( nom4 ) |  |
| tableau=( "element" "${tableau[@]}" ) | Permet d'ajouter un élément au début d'un tableau | tab=( "nom4" "${tab[@]}" ) |  |
| unset nom-tableau | Pour supprimer un tableau | unset tab |  |
| unset nom-tableau[index] | Pour supprimer la case d'un tableau | unset tab[3] |  |
| --- | Tableau associatif | --- | --- |
| ${!tableau[@]} pour l'indice ${tableau[@]} pour la valeur | Tableau uni-dimensionnel dans lequel les indexs sont des chaines de caractères | declare -A eleves=([secondeA]=30 [secondeB]=29)  for i in "${!eleves[@]}"  do  echo "l'indice ici nom de la classe est : $i"  echo "le nombre d'élèves est ici la valeur : ${eleves[$i]}"  done | Il faut utiliser @ (et non \*) pour que la boucle permette d'afficher chacune des valeurs. |
|  |  |  |  |

Fonctions bash

lundi 28 avril 2025

15:26

|  |  |  |  |
| --- | --- | --- | --- |
| Fonction | Description | Exemple | Particularité |
| function nomFonction { suite de commandes } nomFonction | Syntaxe classique pour une commande composée | function f { echo "Bonjour tout le monde" } f | function : c'est la commande function  entre les { } c'est le corps de la fonction  ne pas oublier l'espace avant { |
| nomFonction (){ suite de commandes } nomFonction | Syntaxe sans function | f (){ echo "Bonjour tout le monde" } f | Veiller à bien respecter la casse |
| nomFonction () { suite de commandes } nomFonction | Syntaxe proche du C | f () { echo "Bonjour tout le monde" } f | Une fonction ne peut pas être vide |
| --- | Arguments du script et les paramètres d'une fonction | --- | --- |
| $arg | Argument du script, placés de gauche à droite et du premier au dernier.  la valeur du premier argument est référencée par $1, puis c'est incrémentiel ($2, $3, etc) | function f  {  echo "${0}"  echo "${USER}"  echo "${3}" "${1}" "${2}"  echo "${#}"  echo "${@}"  echo "${\*}"  }  f chez debian facile   #chez : premier argument #debian : deuxième argument  #facile : troisième argument | les paramètres spéciaux $0 (Contient le nom du script tel qu'il a été invoqué), $# (Le nombre de paramètres passés au script), $@ (l'ensemble des arguments, un argument par paramètre) , $\* (l'ensemble des paramètres sous la forme d'un seul argument), $? (Le code retour de la dernière commande), $$ (le PID su shell qui exécute le script), $! (le PID du dernier processus lancé en arrière-plan) |
| "TEST\_NAME : " | Les paramètres de la fonction peuvent être les paramètres passés au script | f (){ echo "SCRIPT\_NAME : " "${0}" echo "USER : " "${USER}" echo "Script argument 1 : " "{$1}" "Script argument 2 : " "${2}" "Script argument 3 : " "${3}" f "${1}" "${2}" "${3}" | Les arguments sont les $1 $2 et $3, ils vont être interprétés par la fonction en elle-meme Penser à spécifier les arguments lors de l'exécution du script : script.sh arg1 arg2 arg3 |
| Shift | Permet de décaler la numérotation des paramètres de position de la fonction | f (){ echo "nom complet : ${0}"  echo " "  echo "${\*}" # avant 'shift 1'  shift 1  echo "${\*}" # après 'shift 1'  echo "${\*}" # avant 'shift 2'  shift 2  echo -e "\t${\*}" # après 'shift 2'  echo -e "\t ${\*}" # avant 'shift 3'  shift 3  echo -e "\t ${\*}" # après 'shift 3'  } | Très important, si c'est printf qui est utilisé à la place de echo, il faut argumenter la tabulation dans les arguments de printf et non dans les arguments de la fonction |
| [redirection] | Permet de rediriger une fonction ( à approfondir) |  |  |

Commandes SSH

jeudi 24 avril 2025

16:55

|  |  |  |  |
| --- | --- | --- | --- |
| Commande | Description | Exemple | Particularité |
| ssh-keygen | Permet de générer une clé publique et privée pour connexion SSH | ssh-keygen -t ed25519 -f ~/.ssh/z420\_ed25519 | -t pour spécifier le type de clé à créer  -f pour spécifier le nom du fichier de clé |
| ssh-copy-id | Permet d'utiliser des clés locales fonctionnelles pour autoriser la connexions sur une machine distante | ssh-copy-id -i ~/.ssh/z420\_ed25519.pub z420.home.arpa | -i pour spécifier l'identify file l'IF doit finir en .pub sinon l'extension est automatiquement ajoutée |
| ssh | Permet de se connecter sur une machine distante | ssh z420.home.arpa | Penser à ajouter en vi dans le ~/.ssh/config les infos en dur pour la connexion.  A savoir : Host, Hostname, User, Port, IdentifyFile  Host z420.home.arpa  Hostname z420.home.arpa  User aicardic  Port 22  IdentifyFile ~/.ssh/z420\_ed25519 |

Commandes Docker

jeudi 24 avril 2025

17:18

|  |  |  |  |
| --- | --- | --- | --- |
| Commande | Description | Exemple | Particularité |
| docker ps | Permet de lister les containers | docker ps -a | -a pour tous les conteneurs |
| docker load | Charge une image depuis une archive .tar ou un STDIN | docker load -i ~/Download/oraclelinux9.tar | -i pour spécifier l'input string |
| docker images | Permet de lister les images, correspond à un ls | docker images |  |
| systemctl start docker | Permet d'activer le service docker | systemctl start docker |  |

Protocoles de tests

**Privilégier les doubles crochets [[**

vendredi 25 avril 2025

08:58

|  |  |  |  |
| --- | --- | --- | --- |
| Protocole | Description | Exemple | Particularité |
| test OP "string"  [OP "string" ]  [[OP "string" ]] | Permet de tester une chaine, vide ou non | test -z  [ -n "string" ]  [[ -z "string" ]]  echo $? pour afficher le code de retour | OP correspond à  -z, vrai si la chaine est de longueur nulle -n, vrai si la chaine est de longueur non nulle echo $? pour fournir le code de retour |
| test string1 OP string2 [string1 OP string2] [[string1 OP string2]] | Permet de comparer deux chaines entre elles | [[ $a == "z\*" ]] # vrai si $a est égal à z\*  [[ $a == z\*]] # vrai si $a commence avec un "z" (reconnaissance de modèle)  [[ "$a" < "$b" ]] vrai si $a se trouve avant $b dans le dictionnaire | OP correspond à  = vrai si string1 correspond à string2 == synonyme de =  != vrai si string1 ne correspond pas à string2  < vrai si string1 est placé lexicographiquement avant string2  > vrai si string1 est placé lexicographiquement après chaine2 |
| test OP file  [OP file]  [[OP file ]] | Permet de tester des fichiers | test -f /tmp/test # teste si le fichier /tmp/test est un fichier  [ -d /temp/test ] # teste si le fichier /tmp/test est un repertoire    echo $? pour afficher le code retour | Sé référer ici pour les primitives de tests : [Image sur la page « Raccourcis clavier/Divers »](onenote:#Raccourcis%20clavier\Divers&section-id={8DFC3A21-C5E9-48F7-B65E-A053DEFE0A5F}&page-id={3232CB37-6B45-4E59-BA31-FDFE1A7FDF53}&object-id={DDFF072B-665D-498C-85B0-E0F7ADA2357A}&10&base-path=C:\Users\aicardic\Documents\Blocs-notes%20OneNote\Mon%20bloc-notes\Linux.one) |
| test int1 OP int2  [int1 OP int2]  [[int1 OP int2]] | Permet de faire des comparaisons numériques | [ 25 -eq 20 ] # teste si 25 est égal à 20 | -eq est égal à  -ne n'est pas égal à  -gt est plus grand que  -ge est plus grand ou égal à  -lt est plus petit que  -le est plus petit ou égal à |
| let int1<OP>int2  let "int1<OP>int2"  (( <int1><OP><int2> )) | Permet de réaliser des comparaisons numériques | (( 42 == 42 ))  echo $? pour afficher le code retour | == vrai est égal != vrai est différent  > vrai si plus grand que  >= vrai si plus grand ou égal que  < vrai si plus petit que  <= vrai si plus petit ou égal que |
|  |  |  |  |

Raccourcis clavier/Divers

vendredi 18 avril 2025

15:41

Texte de remplacement généré par une machine :
RACCOURCI CLAVIER 
Ctrl 1+ 
ACTION 
Déplace le curseur au début de la ligne 
Déplace le curseur à la tin de la ligne 
Efface du curseur à la tin de la ligne 
Efface la ligne jusqu'au curseur. Efface donc la ligne si le curseur se trouve à la tin 
Efface le terminal, équivalent ) çlear 
Effacer du caractère précédent le curseur jusqu'au début du mot 
Comme YU+ w 
Coupe la chaine depuis le caractère situé sous le curseur jusqu'à la tindu mot (si le curseur est placé au début d'un mot, coupe le mot) 
Colle la sélection précédemment coupée 
Inverse les deux caractères précédent le curseur 
Inverse deux mots précédent le curseur 
Met en majuscule la lettre située sous le curseur et déplace le curseur à la fin du mot 
Met en minuscule toutes les lettres depuis la position du curseur jusqu'à la tin du mot 
Met en majuscule toutes les lettres depuis la position du curseur jusqu'à la tin du mot 
Annule la modification précédente 

Texte de remplacement généré par une machine :
CARACTERES 
SIGNIFICATIONS 
Le tilde placé en première position représente le répertoire personnel de l'utilisateur courant. 
Le Collard placé à la tin de l'invite de commande indique le mode user. 
Le dièse placé à la tin de l'invite de commande indique le mode root. 
Le slash sépare les composants d'un seul nom de fichier. 
Début du nom de fichier caché. 
Représente le répertoire actuel 
Début du nom de fichier caché du répertoire actuel. 
Représente le répertoire parent au répertoire actuel 
2) 
A partir de la ligne de commande, appelle le mécanisme d'historique de Basn, ce mécanisme est désactivé dans les scripts. 

Texte de remplacement généré par une machine :
CARACTERES 
$11, 
2 etc 
SIGNIFICATIONS 
Nombre de paramètres 
Paramètre 1, du paramètre 2, etc. 
Nom du programme 
Tous les paramètres de position, vus comme un seul mot( doit être entre guillemets). 
Chacun de tous les paramètres de position (eq à SI S2 etc). 
Fourni le code de retour. 
Le dernier argument de la commande dernière commande. Donc S 
•tficne les options passées au script via 
PID du processus 
PID du processus parent 
PID de la dernière commande lancée en arrière plan 
retour de la sortie standard (attention avec ecno) 

Texte de remplacement généré par une machine :
PRIMITIVES 
-a fichier 
-b fichier 
-c fichier 
-d fichier 
-e fichier 
-t fichier 
-g fichier 
-h fichier 
-k fichier 
-p fichier 
-r fichier 
-s fichier 
-u fichier 
-w fichier 
-x fichier 
-G fichier 
-L fichier 
-N fichier 
-O fichier 
-S fichier 
tichierl 
tichierl 
tichierl 
-et tichier2 
-nt tichier2 
-ot tichier2 
LISTE DES PRIMITIVES DE TEST SUR LES FICHIERS 
RETOURS 
Vrai si fichier existe. 
Vrai si fichier est un fichier spécial bloc. 
Vrai si fichier est un fichier spécial caractère. 
Vrai si fichier existe et est un répertoire. 
Vrai si fichier existe. 
Vrai si fichier est un fichier normal. 
Vrai si fichier a son bit Set-GID positionné. 
Vrai si fichier est un lien symbolique. 
Vrai si fichier a son bit « sticky » positionné. 
Vrai si fichier est un tube nommé (FIFO). 
Vrai si fichier est accessible en lecture. 
Vrai si fichier a une taille strictement positive. 
Vrai si le descripteur de fichier dt est ouvert et se rapporte à un terminal. 
Vrai si fichier a son bit Set-UID positionné. 
Vrai si fichier est accessible en écriture. 
Vrai si fichier est exécutable. 
Vrai si fichier appartient au GID effectif du groupe. 
Vrai si fichier est un lien symbolique. 
Vrai si fichier a été modifié depuis sa dernière lecture. 
Vrai si fichier appartient à l'UID effectif de l'utilisateur. 
Vrai si fichier est une socket_ 
Vrai si tichierl et tichier2 se rapportent au même périphérique et ont les mêmes numéros d'inceuds_ 
Vrai si tichierl est plus récent que tichier2 ou si tichierl existe et non fichier2_ 
Vrai si tichierl est plus ancien que tichier2 ou si tichier2 existe et non tichierl 

Texte de remplacement généré par une machine :
OPERATEURS 
PRIORITES ET ASSOCIATIVITE DES OPERATEURS 
DESIGNATIONS 
ASSOCIATIVITES 
Les priorités sont classé ci-dessous par ordre décroissant. 
Sur une même ligne, les opérateurs ont même priorité. 
C) 
expr++ exp 
++expr -expr 
l) 
11 
Parenthèses 
Post-incrément et post-décrément de variables 
Moins et plus unaires 
Pré-incrément et pré-décrément de variables 
Négations logique et binaire 
Exponentiation (puissance) 
Multiplication, division, modulo 
Addition, soustraction 
Décalage arithmétique à gauche et à droite 
Relationnels 
Egalité et différence 
ET binaire 
OU exclusif binaire 
OU binaire 
ET logique 
OU logique 
Opérateur conditionnel 
Affectations 
Virgule 
Sources : Les priorités proviennent de page du manuel de basn (section EVALUATION ARITHMETIQUE)_ 
Les associativités du langage C. 

Texte de remplacement généré par une machine :
Expressions rationnelles 
Les expressions rationnelles, ou rggep, peuvent remplacer la chaine de caractères dans l'argument du programme grep_ 
Voici des exemples 
EXPRESSION 
abc 
abcS 
nabcS 
nint * main 
VALEUR 
Cherche la chaine abc n'importe où dans la ligne. 
Cherche la chaine abc en début de ligne. 
Cherche la chaine abc en tin de ligne. 
Cherche les ligne ne contenant que abc (commençant et se terminant par abc). 
Cherche n'importe quelle chaine de caractères commençant par st, suivie de deux lettres minuscules et se terminant par ic (i.e static)_ 
Cherche n'importe quelle chaine de caractères en début de ligne commençant par int, suivie de n'importe quoi et se terminant par argc (i.e int main (int argc, char "argv))- 
Cherche n'importe quelle chaine de caractères commençant par X, suivie de O, 1, ou 2 et se terminant par un chiffre (i.e XII). 

Texte de remplacement généré par une machine :
Enchaînements de commandes modifier le 
Enchainements simples [modifier modifier le wikicode] 
Exécutions simultanées [modifier modifier le wikicode] 
com1 & com2 & com3 & 
. & comN 
Les commandes coml jusqu'à comN sont exécutées parallèlement 
Exécutions successives [modifier modifier le wikicode] 
com1 
com2 
comN 
Les commandes coml jusqu'à comN sont exécutées successivement, les unes après les autres 
Enchainements conditionnels modifier le 
Et.. et 
com1 
et [modifier modifier le wikicode ] 
com2 && 
comN 

Notes

mardi 15 avril 2025

09:53

KDE Partition Manager pour le management des partitions sur Linux

Ventoy pour utilitaire de partitions

Pour créer un builde et un iso, Ben a crée un script trouvable dans ~/Git/genOracleLinux9Iso sous le nom de make.sh

ctrl + alt + F2 pour ouvrir changer de tty

ctrl + alt + suppr pour redémarrer

ctrl+maj+v pour copier dans une konsole

ctrl+maj+t pour ouvrir un nouvel onglet dans la console

ctrl+d pour fermer l'onglet courant dans la console, ne marche pas si le prompt n'est pas vide

Le dossier sysconfig dans /etc/ est le concentré de toutes les configurations internes

/etc/passwd contient la liste de tous les utilisateurs

/etc/group continent la liste de tous les groupes

Il y a deux types de sorties linux :

stdout pour la sortie standard

stderr pour la sortie destinée au messages d'erreur

stdr-username.log pour afficher les erreurs de CSLab, fichier contenu dans le $home/GENETTIQ/

~/.bashrc pour modifier le PATH user  
source ~/.bashrc pour update le PATH

/dev/null est un trou noir

Instaurer de mettre toujours en début de script les commandes suivantes :

set -eux

set -o pipefail

ctrl+k ctrl+c pour commenter les lignes sélectionnées

ctrl+k ctrl+u pour retirer le commentaire des lignes sélectionnées

/etc/profile.d/tmout.sh pour modifier le timeout de la console

${BASH\_SOURCE[0]} variable d'env pour donner le path complet du fichier exécuté

%/\* pour manipuler le dossier parent  
%.\* pour retirer les extensions

Appstream et baseOs comme dépôt net à réactiver si désactivation du dépôt dnf "local"