Plan

Unix/Bash

Introduction

Ordinateur

Un modèle en couche

- une couche matériel, une couche système d'exploitation et une couche application

Un ordinateur

Machine programmable servant au traitement de l'information (les données).
Est constitué de :

- Mémoire centrale : stocke le programme en exécution dans la mémoire ainsi que les données en cours de traitement
- Processeur(s) : unité de traitement exécutant les instructions du programme et modifiant les données
- périphériques d'entrée/sortie : tout ce qui permet d'échanger avec l'extérieur : souris, clavier, etc.

Système d'exploitation (SE)

Programme complexe réalisant les fonctionnalités de base indispensables à tous les utilisateurs (interactions avec le système, les périphériques)
Ex. de SE : Windows, MacOS, Linux, DOS, etc.

- SE mono-processus, mono-utilisateur : exécute un programme à la fois avec un seul utilisateur (DOS)
- SE multi-processus, mono-utilisateur : plusieurs tâches simultanées correspondant à des programmes différents. Répartition des ressources à tour de rôle entre les applications. (Windows 95, Android 5)
- SE multi-processus, multi-utilisateurs. Plusieurs utilisateurs peuvent lancer en même temps leurs programmes sur la même machine. (exemple : Unix)

Les stations de travail à l'ENSEIRB-MATMECA sont multi-processus, multi-utilisateurs

processus : toute occurrence en mémoire d'un programme en cours d'exécution

Systèmes de type Unix

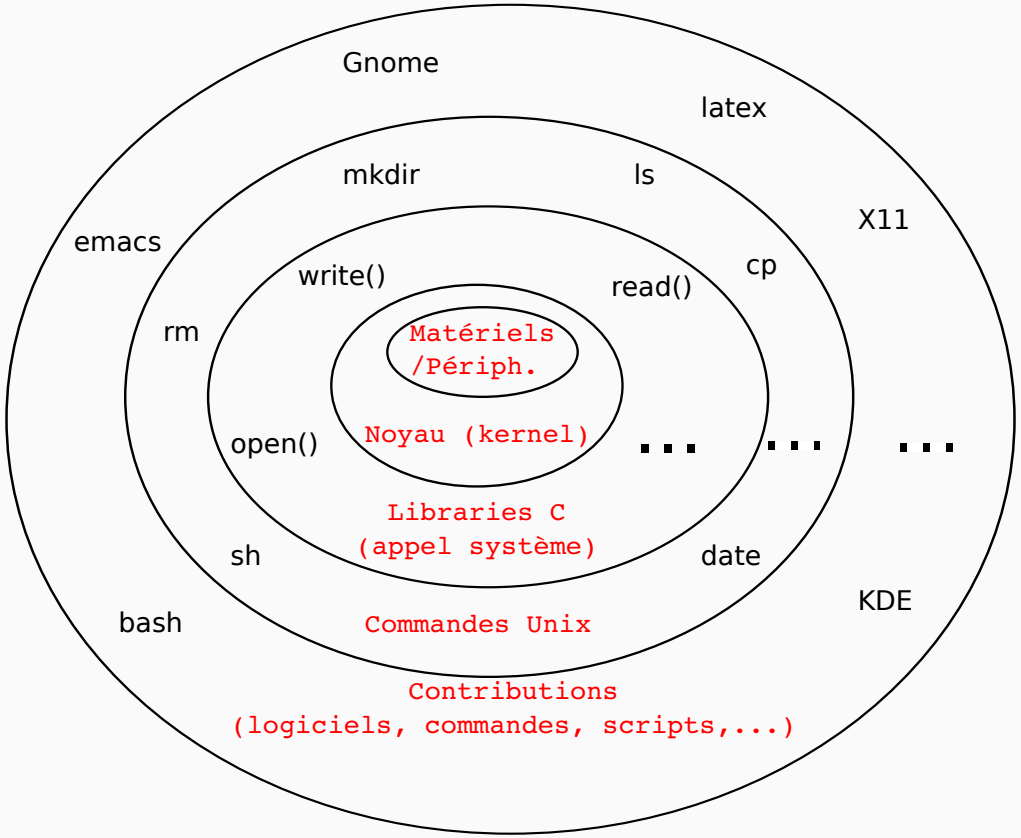
- Plusieurs distributions (versions) par ex : Knoppix, Ubuntu, Linux Mint, Fedora, Gentoo, Debian, ...
- Modèle client/serveur
- Plusieurs terminaux/utilisateurs se connectent à une machine, locale ou distante grâce au réseau

Quelques caractéristiques d'UNIX :

- système à temps partagé : exécution concurrente de processus, ordonnancement
- système multi-utilisateur : compte, connexion (login/password), environnement personnalisé
- partage des ressources : arborescence globale des fichiers, mécanismes de protection
- gestion des communications : entre processus, entre utilisateurs
- simple, portable, extensible

Organisation générale

Un noyau central qui gère : système de fichiers, processus, mémoire, communications, etc. Autour un ensemble de commande et logiciels.



Système de fenêtrage

pas de fusion entre SE et système d'interface graphique (différent de Windows)

- Système de fenêtrage indépendant
- Possibilité de le modifier (KDE, Xfce, Gnome, Unity, ...)
- Possibilité de lancer des fenêtres à distance via la commande `ssh -X`
- Possibilité d'utiliser l'ordinateur même sans interface graphique

Compte utilisateur et protection

- connexion à un système multi-utilisateur : nécessite un compte
- le système connaît la liste des comptes/utilisateurs autorisés
- à chaque compte est associé : un nom **(login)** un mot de passe **(password)** un espace privé **(home directory, ~)**

Rappel sur la protection des comptes/mots de passe

- **Vous ne devez pas** prêter votre compte
- **Vous ne devez pas** prêter votre mot de passe (même pour échanger copier des fichiers)
- **Vous êtes responsable** de ce qui se trouve dans votre compte
- **Vous êtes responsable** de ce que vous faîtes sur le réseau de l'école (logs analysés)
- **Votre mot de passe** doit être complexe (majuscule, minuscule, alpha numérique, etc)
- **Votre mot de passe** doit être résistant à des attaques type brute force ou dictionnaire → pas de noms, pas de mots usuels, ...

Première manipulation

Première connexion

- Démarrer la machine et choisir l'item 1 (démarrage de Fedora), le noyau est chargé en mémoire, c'est la phase de boot
- Un fois terminé la phase de boot : le système d'exploitation démarre et vous obtenez une invite de commande : connectez vous (login/password)
- mot de passe validé via `http://bbb.ipb.fr`
- si ok : le window manager se lance (Gnome), bienvenue sous Linux, vous êtes dans votre *homedirectory*
- chercher dans *Applications/Accessoires* et lancer l'application *Terminal* (interpréteur de commande/console) (Alt-F2)
- vérifier que vous êtes chez vous en tapant la commande `pwd` (`print name of current/working directory`) (vous devez voir quelque chose de la forme `/truc/machin/votrelogin`)
- lancer dans la console le navigateur internet `firefox`
- vérifier que votre connexion à internet fonctionne

Le système de fenêtrage

Quelques commandes **clavier** de base pour bien utiliser son environnement (à remplir)

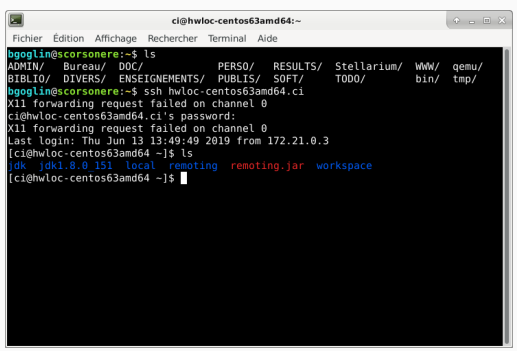
Raccourci	*Fonctionnalité*
	Lancer une application
	Maximiser la fenêtre active
	Minimiser la fenêtre active
	Changer de fenêtre active
	Fermer la fenêtre active
	Se déplacer sur le bureau en dessous
	Déplacer la fenêtre active sur le bureau en dessous
	Déplacer la fenêtre active sur le bureau

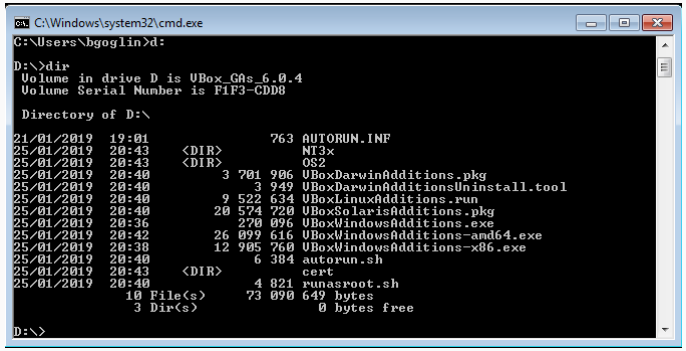
Unix/Bash

Shell - Interpréteur de commandes

Le shell

Le shell est un programme permettant d'interagir avec les services fournis par le système d'exploitation.





- On va va utiliser des programmes dédiés à certaines opérations de base *Déplacer des fichiers, renommer un fichier, changer de dossier, changer des droits, etc.*
- Pas très pratique au premier abord
- Extrêmement puissant avec un peu de pratique
- Quel outil utilises t'on autrement ?

Quelques définitions

- shell : c'est un interpréteur de commandes pour interagir avec le noyau.
  (shell : coquille qui entoure le noyau)
- interpréteur de commande : programme (processus) qui fournit une interface texte entre l'utilisateur et le noyau
  rôle : lire la ligne de commande et exécuter les actions associées
- bash : Bourne-Again shell
  C'est le programme shell par défaut dans la plupart des environnements.

Remarque

Dans la suite du cours, nous mélangerons souvent les termes shell et bash pour désigner la même chose. Mais il existe plusieurs implémentations de shell différentes comme : `sh`, `csh`, `zsh`, `ksh`, `cmd.exe` ...

Le shell - Fonctionnement

Fonctionnement général – le shell exécute la boucle ∞ suivante :

```
POUR toujours FAIRE :
  | Affichage du prompt (prêt à lire) et attente d'une commande
  | Vérification de la syntaxe
  | SI la syntaxe est correcte ALORS :
  |    exécuter la commande
  |    attente de la fin de la commande en cours
  | SINON
  |    afficher un message d'erreur
```

Votre prompt est la chaîne de caractères, appelée également **invite de commande** qui s'affiche dans votre terminal avant que vous ne tapiez une commande.

Traditionnellement, il se termine par un `$` pour un utilisateur normal, et par un `#` quand on est connecté en tant que superutilisateur. On verra plus tard comment le modifier.

Qu'est ce qu'une commande shell ?

- C'est une suite de mot-clés séparés par le caractère de séparation : **l'espace**
  Ne pas mettre d'espace dans les noms de fichiers
- Toute commande Unix a la même syntaxe : (séparateur : l'espace)

```
cmd options arguments
ls  -l      Documents
```

- `enter` lance l'interprétation de la commande
- résultat de la commande dans le terminal
- `options` est suite de mot clés de la forme :
  `-o` ou `-o valeur` pour les options *courtes*
  `--option` ou `--option=valeur` pour les options *longues*
- Remarques :
  - Lors de la description d'une commande, toute option ou paramètre optionnel sera affiché entre crochets `[-v]`, et toute option ou paramètre obligatoire est affiché sans crochets.
  - Souvent, seul le nom de commande est obligatoire.

Quelques commandes de base

Entrer dans le sous-dossier `divers` du sous-dossier `autres` :

```
$ cd autres/divers
```

Savoir dans quel dossier on se trouve :

```
$ pwd
/home/login/autres
```

Lister le contenu du dossier courant :

```
$ ls
dossier1 dossier2 fichier.txt
```

Lister le dossier courant avec les détails :

```
$ ls -la
```

Exercice : premier contact

- ouvrez un terminal
- tapez `echo hello world !`, validez avec entrée
- que s'est-il passé ?
- exécutez maintenant `ls`
- que s'est-il affiché ?
- exécutez `cd <nom>` où vous remplacez <nom> par un des répertoires disponibles
- `ls`
- à quoi sert `cd` ?
- `cd ..`
- `ls`
- que s'est-il passé ?
- `ls --reverse`
- quelle est la différence ?
- `ls -l`
- quelle est la différence ?
- `ls -l <nom>`

Obtenir de l'aide

la commande `man` :

- affiche et formate l'aide associée à la commande, ou fonction, passée en paramètre
- `man <cmd>` : affiche le manuel de `<cmd>`
- `man -h` : affiche une courte aide sur `man`
- `man <n> <com>` : affiche une page spécifique à une section (homonyme)
  → 1 : Programmes exécutables ou commandes de l'interpréteur de commandes
  → 3 : Appels de bibliothèque
  Exemple : `man sleep` et `man 3 sleep`
- `q` pour quitter
  `espace` pour faire défiler d'un écran
  `up/down` pour faire défiler par ligne
  `/<pattern>` pour faire une recherche de
  `n/N` pour le résultat suivant/précédent

Autre : `http://www.explainshell.com/` et
`http://www.grymoire.com/Unix/index.html`

Exercice : manuel

Familiarisons-nous avec `man` :

- `man man`
- Quelles sont les sections des manuels ?
- `man ls`
- quittez le manuel
- `man ls`
- quel est le nom complet de l'option `-a` ?
- quelle est l'option à une lettre équivalente à `--reverse` ?
- quel est le nom complet de l'option `-l` ?

Pour les curieux :
ouvrez le man de `gcc`. Avec la touche espace, essayez d'atteindre le bas du texte.
Bonne chance !

Historique et raccourcis utiles

Le shell intègre des fonctions de gestion de l'historique des commandes exécutées

- C-r (reverse search) : recherche avec motif
- C-p (previous) : commande précédente (up)
- C-n (next) : commande suivante (down)
- history : affiche l'historique des commandes
- !num : relance la commande de numéro donné dans l'historique

Quelques raccourcis utiles pour le shell

- C-k (kill) : coupe et mémorise le texte depuis la position du curseur jusqu'à la fin de la ligne
- C-y (yank) : insère le texte précédemment supprimé
- C-a : début de ligne
- C-e : fin de ligne
- **Tab : active la complétion** **(à utiliser sans modération ! !)**

Exercice : quelques commandes en vrac

- Essayer les commandes suivantes et interpréter les résultats :
  `dite`
  `date`
  `whoami`
  `pws`
  `pwd`
  `cal`
  `cal 2012`
  `cal 10 2012`
  `who`
  `uname -a`
  `uptime`
  `top` (comment quitter cette commande ?)
  `time sleep 5`
  `history`
- Grâce à la complétion rechercher une commande qui nettoie l'écran (`cl...`) vérifier à l'aide du manuel
- Grâce à la complétion rechercher une commande qui permet de quitter le shell (`ex...`). Vérifier à l'aide du manuel

Enchaînement et combinaison de commandes shell

- L'enchaînement de plusieurs commandes se fait avec `;`
  `cmd1; cmd2; cmd3`
- Le groupement de commandes se fait à l'aide des parenthèses
  `(cmd1; cmd2)`
- On peut combiner les commandes en faisant des tests sur les codes de retour :
  `cmd1 || cmd2`
  `cmd1 && cmd2`
- **A ne pas confondre avec :**
  `cmd1 & cmd2`
  `cmd1 | cmd2` (Voir la section redirections)

Exercice : Enchaînement et combinaison de commandes shell

Observez et commentez :

- `cd .. ; ls ; cd; ls ;`
- Essayez de comprendre le comportement des 4 commandes suivantes :
  - `ls .     || ls ..`
  - `ls /toto || ls ..`
  - `ls .     && ls ..`
  - `ls /toto && ls ..`
- A comparer avec `ls /toto; ls ..`
- Plus compliqué. Quelle est la différence avec :
  - `ls .     & ls ..`
  - `ls /toto & ls ..`

  (On lancera plusieurs fois la commande pour bien observer le comportement.
- lancez la commande `firefox`
- est-ce que vous pouvez exécuter une nouvelle commande dans le shell ?
- fermez `firefox` (alt+f4)
- Déduire des questions précédentes une solution qui permette de lancer firefox tout en récupérant le contrôle du terminal.

Gestion des processus

processus : toute occurrence en mémoire d'un programme en cours d'exécution. Il est identifié par un numéro appelé le **PID**

- Lancer une commande en tâche de fond : `cmd &`
- **`C-z` : endort le processus en cours**
- Réveiller un processus endormi en tant que tâche de fond : `bg [n]` (backgroud)
- Reprendre la main sur processus endormi ou en tâche de fond : `fg [n]` (foreground)
- **Attention**, le numero de job donné à fg/bg pour manipuler un processus spécifique n'est pas le *pid*. Il est obtenu avec `jobs`
- Obtenir le *pid* d'un processus :
  `ps`, ou `echo $!` pour la dernière commande lancée.
- **`C-c` : termine le processus en cours**
- `C-d` : quitte le shell en cours (=`exit`)
- Terminer un processus spécifique
  `kill -KILL pid`

Exercice : gestion des tâches 1

- Lancer `firefox`
- terminer le programme par un raccourci clavier et reprendre la main
- relancer la commande ; endormir le processus avec un raccourci clavier pour récupérer la main
- revenez sur firefox et essayer d'interagir avec le navigateur. Est-ce que c'est possible ?
- revenez sur le terminal, et repasser firefox au premier plan
- Endormez de nouveau firefox, puis faites le passer en tâche de fond
- Essayez à nouveau d'interagir avec le navigateur
- Revenez sur le terminal et essayez de tuer le processus de firefox depuis la console.

Exercice : gestion des tâches 2

- Lancez la commande `ls -l&`
- Que se passe-t-il ? Pouvez vous l'expliquez ?
- Tapez la commande `firefox & xeyes &`
- Reprenez la main sur firefox en premier plan
- Terminez firefox avec un raccourci clavier
- Lancez `firefox`, puis endormez le
- Lancez `xeyes`, puis endormez le
- Faites exécutez firefox en arrière plan, sans modifier l'état de xeyes.

Unix/Bash

Arborescence et système de fichiers

Système de fichiers

Les fichiers sont structurés sous la forme d'un arbre dont la racine est `/` :

- les noeuds sont des répertoires
- les feuilles sont des fichiers *(au sens large)*

Petit lexique :

- fichier : file
- répertoire : directory
- chemin : path
- répertoire de connexion : home directory (`~`, *tilde*)
- répertoire courant : working directory (`pwd`)

Particularité : sous Unix tout peut être vu comme un fichier

- fichier régulier : fichier contenant du texte, du code source, des commandes exécutables, ...
- répertoire : un fichier contenant d'autres fichiers
- fichiers spéciaux : liens vers des périphériques, canaux de communication entre processus

Dans la suite, si on ne précise pas fichier *régulier*, c'est donc qu'il s'agit de fichier au sens générique désignant n'importe quel noeud de l'arbre (noeud ou feuille).

Notion de chemin

Un chemin identifie un fichier dans le système.

- Il correspond à un déplacement dans l'arborescence des fichiers.
- Il s'écrit comme une suite de noms entrecoupés de `/`
  `/usr/share/fonts`
- On distingue deux parties dans un chemin :
  - le **`dirname`**, ou préfix, qui précède le dernier `/` du chemin
    Le préfix est **toujours** le chemin d'un répertoire
    `/usr/share`
  - le **`basename`** qui suit le dernier `/` du chemin

    `fonts`
- Un chemin est valide si son préfixe est le chemin d'un répertoire existant et si le basename est le nom d'un fichier existant

Deux noms sont réservés dans l'arborescence de fichier et sont contenus dans tout répertoire R sous Unix :

- un élément `'.'` : référence le répertoire R
- un élément `'..'` : référence le répertoire père de R

D'où le `cd ..` dans l'exercice page 15.

Notion de chemin

Un chemin comme son nom l'indique correspond à un déplacement d'un point A à un point B. Il est donc nécessaire de déterminer le point de départ.

Chemins absolus

Le point de départ **ne dépend pas de la position courante** (working directory) dans le système de fichier. Il est défini quelque soit l'endroit où on se trouve dans l'arborescence. Il commence donc par :

- `/` : La racine du système de fichier
- `~` : Alias/Raccourci vers le répertoire home de l'utilisateur courant
- `~toto` : Alias/Raccourci vers le répertoire home de l'utilisateur `toto`

Chemins relatifs

Le point de départ **dépend de la position courante** (working directory) dans le système de fichier. Il n'est défini et valide que pour la position courante dans l'arborescence. Il commence par :

- `./` : Le répertoire courant
- `../` : Le répertoire parent
- `subdir1` : un fichier contenu dans le répertoire courant (équivalent à `./subdir`

Notion de chemin

Soit l'arborescence suivante :

```
/usr/
|--local/
|   |--bin/
|   |   |--emacs
|   |--lib/
|   |   |--libm.so
```

Si le répertoire courant est `/usr/local/` :

- On peut faire référence au sous-répertoire `bin` avec :
  `../local/bin`, `bin`, `./bin`, ...
  `/usr/local/bin`, `/usr/local/../local/bin`, ...
- On peut faire référence au fichier `emacs` avec :
  `bin/emacs`, `../../usr/../usr/local/bin/emacs`, ...
  `/usr/local/bin/emacs`, `/usr/local/./bin/emacs`, ...

Arborescence standard

Dans la majorité des cas (variantes possibles selon les versions)

- les répertoires de fichiers exécutables : `/bin, /usr/bin, /usr/local/bin`
- répertoires de bibliothèques : `/lib, /usr/lib, /usr/local/lib`
- pages de manuel : `/usr/man`
- répertoire relatif à l'administration `/etc` ex. `/etc/passwd` (contenant des infos sur les utilisateurs)
- divers : `/tmp`, `/dev` (fichiers spéciaux : drivers)

Chaque processus lancé possède un répertoire courant : sa position dans l'arborescence

- Ex. lors la connexion, le répertoire courant du shell est le répertoire de l'utilisateur (home directory ou `~`)
- Pour modifier cette position courante on utilise la commande : `cd` (change directory)

Quelques règles pour le nommage des fichiers

- pas d'espaces : utiliser "_" (underscore) ou "-" (minus)
  espace est le caractère de séparation des mot-clés des commandes bash
- pas de caractères spéciaux : pouvant être interprété par le shell ($, , ...)
- ajouter un suffixe pour désigner les types de fichiers : .txt, .html, .sh, .tgz, ...
- **L'extension ne défini pas le type de fichier ! ! !**
  le nom d'un pdf peut très bien se terminer par «.txt», mais c'est pas une raison pour faire n'importe quoi.
- Utiliser la commande `file` pour connaître le type d'un fichier

Commandes de manipulation de fichiers

- `cd` (change the working directory)
- `pwd` (print name of current/working directory)
- `ls`, -l, -la, -G (list directory contents)
- `cp`, -r (copy files and directories)
- `mkdir` (make directories)
- `mv` (move (rename) files)
- `rm`, -r (remove files or directories)
- `touch` (change file timestamps) si le fichier n'existe pas création d'un fichier vide
- `file` (determine file type)
- `cat` (concatenate files and print on the standard output)
- `less` affiche un fichier avec possibilité de le parcourir
- `wc`, -l, -w, (print newline, word, and byte counts for each file)
- `chmod` (change file mode bits) changer les droits utilisateurs. Commande très importante, dont nous allons parler un peu plus loin.

Exercice : échauffement des doigts

Essayer la séquence de commandes suivantes et donner/interpréter les résultats :

- `cd`
- `pwd`
- `ls -la`
- `cd .`
- `pwd`
- `cd ..`
- `pwd`
- `ls -la`
- `cd ..`
- `pwd`
- `cd /etc`
- `ls -la`
- `cat passwd`
- `cd -`
- `cd ..`
- `pwd`
- `cd ..`
- `pwd`
- `cd`
- `pwd`

Exercice : manipulation des fichiers

Après chacune des commandes suivantes, vérifier que la commande a été validée grâce à la commande `ls`

- Déplacez-vous dans votre home directory
- Créer un répertoire nommé `subdir`
- Copier le fichier `/etc/passwd` dans `subdir`
- Copier le fichier `/etc/passwd` dans `subdir` en le nommant `mypasswd`
- Allez dans le répertoire `subdir`
- Vérifier la présence des fichiers `passwd` et `mypasswd`
- Renommer le fichier `passwd` en `passwd.bak`
- Afficher le contenu de ce `passwd.bak`
- Remonter au répertoire père de `subdir`
- Copier `subdir` en `subdir.bak`
- Supprimer `subdir`
- Créer le fichier `test` à l'intérieur de `subdir.bak`
- Gardez votre dossier pour l'instant, nous nous en servirons encore une fois.

Droits des fichiers

Toute opération sur un fichier est soumise à un ensemble de droits d'accès qui sont de 3 types.

r : droit de lecture

- Si répertoire, consultation de ses entrées (c.-à-.d, `ls` autorisé)
- Sinon, consultation du contenu du fichier

w : droit d'écriture

- Si répertoire, droit de création, de renommage et de suppression d'une entrée dans le répertoire
- Sinon, droit de modification du contenu du fichier

x :

- si répertoire, droit de traverser (c.-à-.d., `cd` autorisé)
- sinon, droit d'exécution

Droits des fichiers

- 3 catégories d'utilisateurs :
  - Propriétaire (`u`)
  - Groupe propriétaire (`g`)
  - Tous les autres (`o`)
- Chaque catégorie possède ses types d'accès `r w x`

Droits des fichiers - consultation

- `ls -ld` → donne les droits des fichiers
- Format de sortie de `ls -l`
- `- --- --- ---`
- `-` → Type du fichier
  - `-` : fichier régulier
  - `d` : répertoire
  - `l` : lien symbolique
  - `s` : socket
  - ...
- `---` → Droits du propriétaire (rwx)
- `---` → Droits du groupe (rwx)
- `---` → Droits des autres (rwx)

Droits des fichiers - modification

- Modification des droits sur un fichier/répertoire existant
  - `chmod droit fichier`
- Droits à appliquer :
  - Catégorie : u (=user), g (=group), o(=other), ou a (=all)
  - Opérations : + (ajout), - (retrait), = (affectation)
  - Example : u+rx,o=r
- Alternativement, on peut aussi les donner en octal : 4 (lecture), 2 (écriture) et 1 (exécution)
  Exemple : 641 pour rw-r—-x, 752 pour rwxr-x-w-

Exercices

1.
   - Expliquer `-rw-------`
   - Expliquer `drwx--x--x`
   - Afficher le contenu du fichier `~mfaverge/tmp/file.txt`
2.
   - A qui appartient le fichier `/etc/passwd` ?
   - Pourquoi n'avez vous pas le droit de le modifier ?
   - Quels sont vos droits sur ce fichier
3. Donner deux façons possible d'obtenir les droits suivants (en symbolique et en octal)
   - `-rw-r--r--`
   - `-rwxr-xr-x`
   - `-rwxr-xr--`
   - `-r--r--r--`
   - `-r----x---`

Exercices

4. Soit le répertoire `drwxr-xr-x 2 user staff 68B Mar 23 12:01 tmp` Donner et expliquer le résultat des commandes suivantes sur `tmp` en supposant que `tmp` ce trouve dans le homedir de `user`
   - `chmod 355 tmp; ls tmp`
   - `cd tmp; pwd`
   - `cp /etc/passwd .; ls`
   - `cd ..; chmod 755 tmp; ls tmp`
   - `chmod 655 tmp; cd tmp`
   - `chmod 555 tmp; rm -f tmp/passwd`

   Faîtes les modifications nécessaires jusqu'à suppression de ce répertoire.

Exercices

1. Revenez à votre `subdir.bak` (`cd  /subdir.bak`).
2. Enlever le bit de lecture sur le fichier `passwd.bak` et vérifier à nouveau les droits
3. Afficher le contenu de ce `passwd.bak`
4. Enlever le bit autorisant le parcours sur le répertoire `subdir.bak`
5. Supprimer le répertoire `subdir.bak`

Travail à distance

Connexion sécurisée à une machine distante

- `ssh [-X] login@hostname` (secure shell)
- `-X` permet d'accéder aux applications graphiques
- `hostname` est le nom de la machine sur laquelle se connecter
- A l'ENSEIRB : ssh.enseirb-matmeca.fr (passerelle), puis travail64 ou toute autre machine (babaorum, tartopum, lsd...)

Copier un fichier vers une machine distante (secure copy)

- `scp [-r] src1 [src2 ... srcN] login@hostname:dst_path`
- `dst_path` est un chemin absolu, ou un chemin relatif à partir de `~login`
- Suit les mêmes règles que `cp`

Copier depuis une machine distante vers la machine locale

- `scp login@hostname:src_path dst_path`
- Suit les mêmes règles que `cp`
- 1 seul fichier (au sens large) peut être la source

Utilisation avancée

- Utilisation de clés : ssh-keygen
- Configuration du client ssh (`man ssh_config`)

Unix/Bash

Substitutions et variables

Substitutions et expansions (1/3)

Le shell analyse la ligne de commande avant de l'exécuter pour faire des substitutions/expansions. L'ordre des substitutions est toujours le même, il y a entre autres :

- **Expansion des accolades** :
  `a{b,c,d}e` = `abe ace ade`
- **Expansion des tildes ~**
  `~` → chemin vers votre homedir
  `~toto` → chemin vers le homedir de toto
- **Expansion des variables, variables d'environnement** :
  - syntaxe d'une variable : `$VAR`, `${VAR}`, `${VAR:-defaut}`
  - affectation d'une variable :
    `export VAR=value` → variable disponible dans l'environnement des processus fils
    `VAR=value` → variable disponible **uniquement** dans l'environnement du processus courant

    Attention PAS d'espace entre le nom de variable et le =

Substitutions et expansions (2/3)

- **Substitution de commandes :**
  `${cmd}` exécute `cmd` dans un sous-shell et remplace `${cmd}` par le résultat de l'exécution de la commande.
- **Coupures des mots :**
  - Chaque séquence de blancs (espaces, tabulations, ou retours de ligne) délimite chaque mot-clés (variables, commande, paramètre, option) de la ligne de commande
  - `'une chaîne de caractère'` : → Le contenu n'est pas interprété
  - `"une chaîne de caractère"` : → Le contenu est interprété

Substitutions et expansions (3/3)

Expansion des noms de fichiers

- Analyse de la ligne de commande et recherche des caractères : `*`, `?`, `[`, `]`
- `*` signifie n'importe quelle chaîne de caractère, même la chaîne vide ;
- `?` signifie n'importe quel caractère ;
- `[...]` signifie n'importe quel caractère spécifié entre les crochets qui est représenté comme une suite de
  - caractère seul
  - d'ensemble de caractère séparé par un - (Notez que si - est le premier carcatère de l'ensemble il s'agit du caractère - et non d'un ensemble)

  Exemple : `[04a-dG-L6]` représente l'ensemble des caractères composé de : $0, 4, a, b, c, d, G, H, I, J, K, L, 6$

Le caractère d'échappement \ permet de protéger les caractères spéciaux précédent lorsque qu'ils ne doivent pas être évalués.
Par exemple : `mkdir my\ dir`

Rappel : **Utilisez et abusez de la touche TAB pour la complétion !**

Exercices

- Que fait `echo projet/{src/{module1,module2},lib,bin}` ?
- Que produit la séquence suivante ?
  ```
  echo ${BROL:-Je ne le connais pas}
  export BROL="Monsieur Brol"
  echo ${BROL:-Je ne le connais pas}
  ```
- Que produit `echo $PATH` ? Que représente cette variable d'environnement ?
- Que fait la séquence suivante : `export $VAR=$(date); echo $VAR` ?
- Que fait la séquence suivante : `export BROL="Monsieur Brol"`
  ```
  echo "$BROL et trol"
  echo '$BROL et trol'
  ```
- Que fait `ls -l /usr/lib/lib[a-jt-z]??e*`
- Que fait la séquence suivante : `RAC=/bin`
  ```
  echo $RAC
  echo $RAC/pwd
  $RAC/pwd
  ```

Exercices

- Que fait la séquence suivante : `echo /bin/[a-c]?`
- Que fait la séquence suivante : `echo '/bin/[a-c]?'`
  ```
  N='/bin/[a-c]?'
  echo $N
  ```
- Que fait la séquence suivante : `A=un`
  ```
  bash; echo /$A/
  ^D
  export A; bash; echo /$A/
  A=deux; B=trois; echo /$A/$B
  ^D
  echo /$A/$B
  ```

Unix/Bash

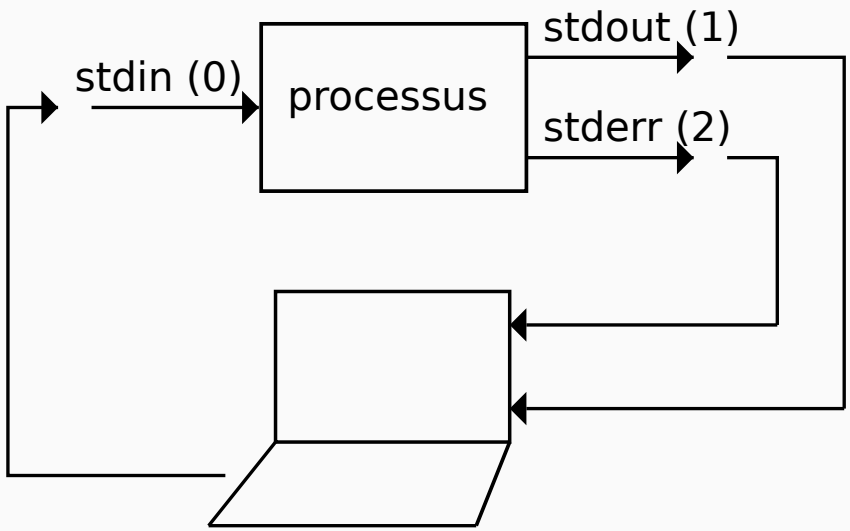
Mécanismes d'entrées/sorties

Mécanismes d'entrées/sorties

Généralement une opération d'E/S :

- transfert de données entre une zone mémoire et un fichier ou un périphérique
- Ex. un processus effectue des E/S : depuis le clavier vers l'écran

A sa création chaque processus possède 3 flots E/S standards

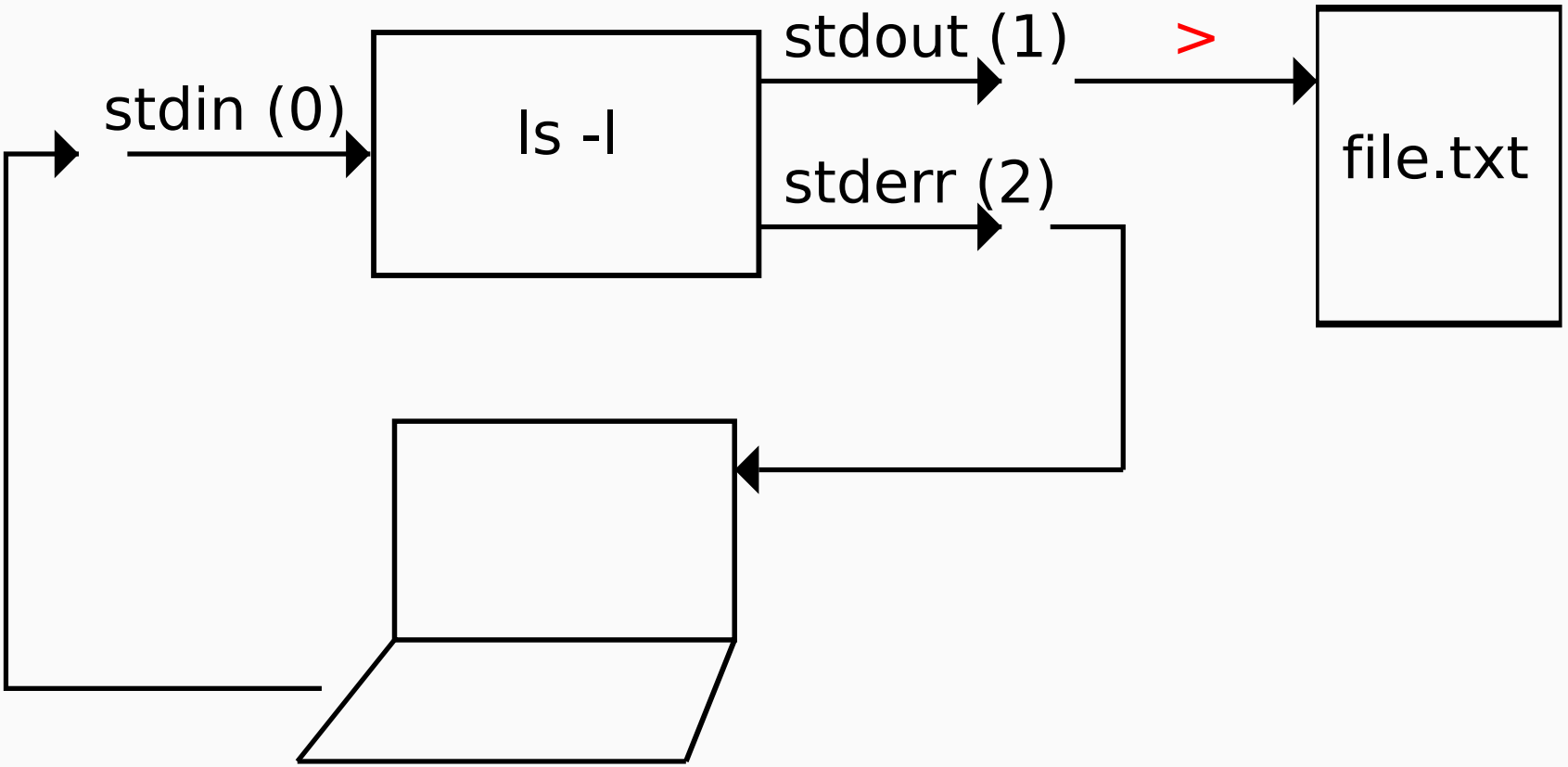


- `stdin (n° 0)` : entrée standard opérations lecture, par défaut le clavier
- `stdout (n° 1)` : sortie standard opérations d'écriture, par défaut l'écran
- `stderr (n° 2)` sortie d'erreur opérations d'affichage des messages d'erreurs, par défaut l'écran

Unix, permet à l'utilisateur de modifier et de reconnecter les flots d'E/S
c'est le mécanisme de redirection

Mécanismes de redirection (1)

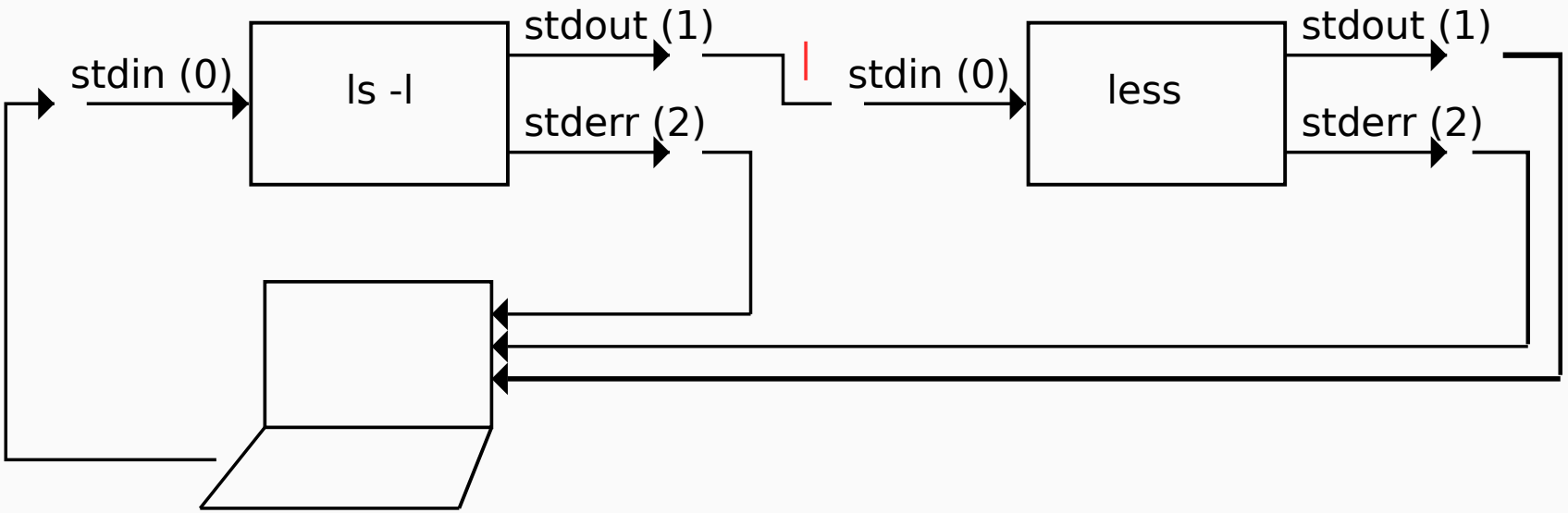
Rediriger un processus (programme) vers un fichier via '>'.
Ex : `ls -l > file.txt`



Si le fichier existe il est écrasé

Mécanismes de redirection (2)

Rediriger un processus (programme) vers un autre processus (programme) via le `pipe` (tube) : `'|'`
Ex. `ls -l | less`



Possibilité de créer des pipelines complexes en une seule ligne de commande

Mécanismes de redirection (3)

D'autres mots clés existent : les plus communs

- `< file` : redirection de l'entrée standard dans un fichier
- `> file` : redirection de la sortie standard dans un fichier (déjà vu)
- `>>` : idem, avec positionnement à la fin du fichier s'il existe
- `cmd | cmd` : le pipe (déjà vu)
- `2 > file` : redirige la sortie d'erreur vers `file`
- `2 >& 1` : fusionne la sortie d'erreur sur le sortie standard

Exercices

1. Que fait la ligne de commande
   `date > tmp.txt; ls -l  /.??* | wc -l >> tmp.txt.`
   Que contient finalement `tmp.txt`
2. Que fait la ligne de commande
   `echo $(date) : $(who | wc -l) util. sur $(hostname)`
3. Rediriger la sortie de `ls -la` vers un fichier
4. Placez vous dans votre `homedir`
   - Déplacez vous dans le répertoire `dir`, un message d'erreur est-il apparu ? si oui où ?
   - Recommencer mais en redirigeant la sortie d'erreur vers le fichier `err`.
   - Un message est-il apparu ? le fichier `err` a-t-il été créé ? Que contient le fichier `err` ?
   - Recommencer mais cette fois en positionnant le curseur à la fin (`>>`). Que contient le fichier `err` ?

University of Minnesota
Supercomputing Institute
for Digital Simulation and Advanced Computation
1200 Washington Ave S • Minneapolis, MN 55415 • Tel: (612)625-1818

UNIX Reference Card

Warnings!!

When a file has been **DELETED** it can only be restored from a backup. The original is gone!

When a file is **OVERWRITTEN** it has been changed forever! It can only be restored from a backup.

Directory Abbreviations

~	home directory (tilde)
~*username*	another user's home directory
.	current working directory
..	parent of current working directory

Communication

ssh [*options*] hostname
Ssh (Secure Shell) a program for logging into a remote host. Replaces telnet, rlogin, and rsh
options:
-l *login_name* — specifies the user to log in on the remote machine

scp [*options*] user@host1:file1 user@host2:file2
Secure copy files between hosts on a network; uses ssh for data transfer.
options:
-p preserve modification times
-r recursively copy entire directories

Comparison

diff [*options*] *file1 file2*
Compare two text files.
options:
-a treat all files as text files
-b ignore repeating blanks and end-of-line blanks; treat successive blanks as one
-i ignore case in text comparison
-q output only whether files differ

File Management

cat [*options*] [*files*]
Read one or more *files* and print them on standard output. Use the > operator to combine several files into a new file; use >> to append files to an existing file.
options:
-n print the number of the output line to the line's left
-s squeeze out extra blank lines

cd [*dir*]
Change working directory to dir; default is the users home directory.

chgrp *newgroup files*
Change the group of one or more *files* to *newgroup*. *newgroup* is either a group ID number of a group name. Only the owner can change the group.
options:
-c print information about those files that are affected
-R recursively apply changes to subdirectories

chmod [*options*] *mode files*
Change the access *mode* (permissions) of one or more *files*. Only the owner of a file or a privileged user may change its mode.
options:
-c print information about affected files
-R recursively apply changes to subdirectories
mode:
can be numeric
4 read
2 write
1 execute
or an expression of the form *who opcode permission*. *who* is optional (if missing, default is **a**)
who
u user
g group
o other
a all (default)
opcode
+ add permission
– remove permission
= assign permission
permission
r read
w write
x execute
X set execution permission only if executable by user

cp [*options*] *file1 file2*
cp [*options*] *files directory*
Copy *file1* to *file2*, or copy one or more *files* to the same names under *directory*.
options:
-a preserves attributes of original files
-f remove existing files in the destination
-i prompt before overwriting destination files
-r recursively copy directories
-s make symbolic links instead of copying

file [*options*] *files*
Classify the named *files* according to the type of data they contain.

less [*options*] [*filename*]
A program for browsing or paging through files or other output. Can use arrow keys for scrolling forward or backward.
options:
see **man** pages for options (type: **man less**)

ln [*options*] *sourcename* [*destname*]
ln [*options*] *sourcenames destdirectory*
Create links for files, allowing them to be accessed by different names.
options:
-b backup files before removing originals
-i prompt for permission before removing files
-s create a symbolic link. This lets you see the name of the link when you run **ls -l** (otherwise there is now way to know the name that a file is linked to).

ls [*options*] [*names*]
List the contents of a directory. If no *names* are given, the files in the current directory are listed.
options:
-a list all files, including hidden files
-c list files by status change time
-l long format listing (permissions, owner, size, modification time)

mkdir [*options*] *directories*
Create one or more directories.
options:
-m *mode* set the access *mode* for new directories. See **chmod** for *mode* formats.
-p create intervening parent directories if they don't exist

more [*options*] [*files*]
Display the content of the named *files* one screen at a time. See less for an alternative.
options:
see **man** pages for options (type: **man more**)

pwd
Print the full pathname of the current working directory.

scp [*options*] user@host1:file1 user@host2:file2
Secure copy files between hosts on a network; uses ssh for data transfer.
options:
-p preserve modification times
-r recursively copy entire directories

mv [*options*] *sources target*
Move or rename files and directories. The *source* and *target* determine the result.

source	*target*	*result*
file	*name*	rename file as *name*
file	existing file	overwrite existing file with source file
directory	*name*	rename directory as *name*
directory	existing	move directory to be a
directory	subdirectory of existing directory	

options:
-b back up files before moving
-f force the move
-i query user before removing files

rm [*options*] *files*
Delete one or more *files*. ***Once a file or directory has been removed it can only be retrieved from a backup!***
options:
-d remove directories, even if they are not empty
-f remove files without prompting
-i prompt for file removal
-r recursively remove an entire directory and its contents, including subdirectories. ***Be very careful with this option.***

Miscellaneous

! Repeat the last command

!*string* Repeat the last command beginning with *string*.

cal [*-jy*] [[*month*] *year*]
Print a 12-month calendar for the given *year* or a one-month calendar of the given *month* and year. No arguments, print a calendar for the current month.
options:
-j display Julian dates
-y display entire current year

clear
Clear the terminal display

history
Display list of most recently executed commands

kill [*option*] *IDs*
Send a signal to terminate one or more process *IDs*.
options:
-l list all signals
-signal the signal number (from **ps -f**) or name (from **kill -l**). You can kill just about any process with a signal number of 9.

man *command*
Display information from the online reference manuals.

jobs [*options*] *job_id*
Display status of jobs in the current session. Simply specifying jobs returns the status of all stopped jobs, running background jobs, and all suspended jobs.
options:
-l provide more information about each job listed
-p display only the process IDs for the process group leaders of the selected jobs

whereis *command*
Locate a *command*; display the full pathname for the *command*.

which [*commands*]
List which files would be executed if the named *commands* had been run.

Searching

egrep [*options*] [*regexp*] [*files*]
grep [*options*] [*regexp*] [*files*]
Search one or more *file*s for lines that match a regular expression *regexp*. To include characters such as +, ?, |, ( ), blank spaces, etc. enclose these expressions in quotes. See **man** pages for the differences between **egrep**, **fgrep**, and **grep**.
options:
-c print only a count of matched lines
-i ignore case
-l list filenames but not matched lines
-n print lines and their line numbers
-v print all lines that do not match *regexp*

find [*pathnames*] [*conditions*]
Useful for finding particular files. **find** descends the directory tree beginning at each *pathname* and locates files that meet the specified *conditions*.
options:
-name *pattern* find files whose name matches *pattern*
-print print the matching files and directories using their full pathnames
see **man** pages for options (type: **man find**)

Storage

compress [*options*] [*files*] – compress file
uncompress [*options*] [*files*] – uncompress compressed file
compress reduces the size of the named *files*. When possible the resulting compressed file will have the file extension **.Z**. Compressed files can be restored using **uncompress**.
options:
-d uncompress file, same as **uncompress**
-v prints the percentage reduction
-V prints the version of compress

gzip [*options*] [*files*] – compress file
gunzip [*options*] [*files*] – uncompress gzipped file
GNU compression utility. Renames compressed files *filename.gz*. Uncompress with **gunzip**.
options:
-d uncompress file, same as **gunzip**
-r recursively compress or decompress files within a directory
-v print name and percent size reduction for each file

tar [*options*] [*tarfile*] [*other-files*]
Copy *files* to or restore *files* from an archive. If any files are directories, **tar** acts on the entire subtree.
options:
-c create a new archive
-d compare the files stored in tarfile with other-files
-r append other-files to the end of an existing archive
-t print the names of files in archive
-v verbose, print filenames as they are added or extracted
-x extract *other-files* from archive, or extract all files if *other-files* not specified

System Status

Control-C
Stop (interrupt) job running in the foreground

Control-Z
Suspend job running in the foreground

date [*options*] [*+format*] [*date*]
Print the current date and time. You may specify a display *format*.
options:
see **man** pages for options (type: **man date**)

df [*options*] [*name*]
Report the amount of free disk space available on all mounted file systems or on a given *name*.
options:
-k print sizes in kilobytes

du [*options*] [*directories*]
Print disk used by each named directory and its subdirectories.
options:
-k print sizes in kilobytes
-s print only the grand total for each directory

env [*option*] [*variable=value ...*] [*command*]
Display the current environment or, if an environment *variable* is specified, set it to a new *value* and display the modified environment.
option:
-u unset the specified *variable*

ps [*options*]
Report on active processes.
options:
-a list all processes except processes not associated with the terminal
-e list all processes
-l produce long format listing
-u *list* list for usernames in *list*

quota [*option*]
Display disk usage and limits
option:
-v report quotas even if they haven't been exceeded

Contact Information

Phone: 612-626-0802

Email: help@msi.umn.edu

WWW: http://www.msi.umn.edu

http://www.msi.umn.edu/user_support