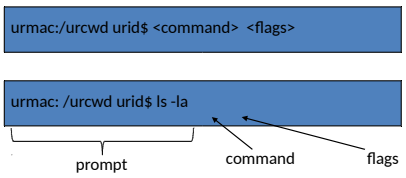
## Unix File Hierarchy

/ – root. The source of everything
/bin – Where executables (programs) are run
/sbin – generally of interest only to the super user
/home – where your "home directories" are
/tmp – same as c:\temp in windows - for temporary files
/var – system logs etc. (probably not interesting)
/usr – laid out like /, but with less system-critical stuff
/dev – -files that represent devices on the system
/proc – has runtime system info (try: cat /proc/cpuinfo)

## Command Notation

- command name
- **[ ]** optional arguments or options
  - can be nested
    - [ arg1 [ arg2 ] ]
    - […]
- **|** is an OR condition
  - arg1 | arg2

## What is a Command




urmac = your machine
urcwd = your current working directory path
urid = your user id.

- Commands are the way to "do things" in unix (no point and click)
- A command consists of a program name and options called "flags"
- Cryptic commands are good for you; they improve typing speed
- Also, after you type them long enough, you get to like them!

## Command info…

- **man** - manual pages
- **info** - variation
- **apropos** searchterm
  - what have you got on searchterm
  - same as man -k
- **whatis** - one line summary

## Command Components

- Commands consist of:
  - Command name
  - Options, or flags
  - Arguments
- Arguments are the "things" that the command will operate on
- Options modify the behavior of the command

## Two Basic Commands

- The most useful commands you'll ever learn:
  - **man** (short for "*man*ual")
  - **info**
- They help you find information about other commands
  - **man** <cmd> or **info** <cmd> retrieves detailed information about <cmd>
  - **man -k** <keyword> searches the man page summaries (faster, and will probably give better results)
  - **man -K** <keyword> searches the full text of the man pages
  - **man man** manual page on the manual command i.e. help on using man

```
$ man -k password
passwd   (5)   - password file
xlock    (1)   - Locks the local X display
          until a password is entered
urmc:/filepath$ passwd
```

## MANual pages - man

Read online documentation
or find reference pages by keyword
Syntax:
- man [[section] title]

or:
- man [[-k keyword] | [-f filename]]

Examples:

urmac % *man ps*
urmac% *man -k compile*
urmac % *man -f /var/spool/mail*
urmac % *man link*

## Stopping a scroll roll & other things

Most modern systems are configured to use a pager to output a page at a time, so it doesn't all scroll by but it is good to know the old handy standbys

**man**

- Output
  - ^s - stop
  - ^w - write
- Input
  - ^d - end of file marker

## Two Basic Commands (info)

- Info, as opposed to man, is category based
- Documents are hyperlinked
  - `info <cmd>` retrieves detailed information about `<cmd>`
  - `info` by itself will give instructions on its usage
  - Type `q` to quit.
  - Type `h` for help.

## Pipe to page More or less!

- The output scrolls forever quickly unless you put it through a basic formatter… …simplest is to use a pipe command… …where the output of one cmd / process… …is fed directly into the input to another
  - (…well via temporary files which the system creates & clears up afterwards automatically)
- man ls | more
- man ls | less
- man ls | pg

## Scrolling around

Output from man etc. is sent through a page formatter with these commands

- space or f or - forward to next page
- b - back a page
- p - to first page
  - (not to be confused with -P option to specify the default pager which is to be used)
- Also use of the arrow keys will control page scrolling

## Pipes for …

Pipes

- 'pipe' output from one command or process
  - Through intermediate files
    - Which the system automatically
      - Creates so they can be used, and saves user doing it
      - Clears up after use to save space
- To another command or process
- And so on…
- *cat myfile* | *grep key* | *sort* | *lpr*
  - From the end:-

    print a sorted list of lines containing key in myfile;

    grep does global regular expression pattern matching

## Typical

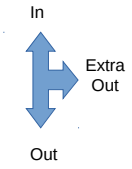
- ls -alR | grep searchterm
  - ls - alR will
    - List all files in long format Recursively from cwd
  - Piping the output to grep which
    - Will only output lines matching the searchterm
- e.g. ls -al | grep music
  - Will output all filenames from ls within the current directory which have music in their title
  - Saves waiting on scrolling and searching through huge output on screen
- **grep** can use regular expressions to specify text patterns
- **find** is a much more powerful prewritten file find cmd for some uses, but for others ls | grep is more flexible…DIY!

## Commands - Basic file

- ls - lists your files
- mv - moves a file
  - can be used to move a file (incl. a directory) into another directory
  - can be used to rename a file within the current directory
- rm - remove a file
- cp - copies a file - makes a real copy (but not of soft links if the file is a directory!)
- ln f1 f2
  - creates hard links to a **file inode** from f1 to f2's inode
  - makes a virtual copy of the file
  - just another name to point to the file's inode
- ln -s - create a soft link to a **filename**, from f1 to f2.

## Tee (join – like plumbing T-piece) e.g.to save & see

- Tee command - as in T-juntion - 2 ways
  - Copies std input to std output, and to named file(s)
  - lets you see the output on screen
  - And can pipe a copy to the next command
  - Functions like pipe, but placed differently
- Tee eeteemessagefile | mailx eeteehome
  - Takes standard input until ^d - just like cat
  - saves a copy in eeteemessagefile
  - and mails the message to eeteehome
- Ls -alR | grep music tee musicinfilename
  - Lists all files long format recursively,
  - Searching for filenames with 'music' in their name
  - Writing the output to the screen and to a file musicinfilename, creating it if it is not there, and overwriting it if it is
  - Use tee >>musicinfilename to append output to end of file




## Commands - Basic file

- Inodes
  - all files are accessed throught their inodes
  - a data structure with a unique id within a single filesystem
  - which uniquely identifies and points to the file
  - and contains other admin data on the file
- Note the second column of the ls command after rwx modes show how many links a file has
- Links
  - Hard - link to a file's inode
    - file not deleted until all hard links are deleted
    - inode & file is not lost if 'spare' extra link to file is moved, copied or deleted
  - Soft - link to a file's name – which is the only link to the inode
    - is lost if original file (name) is moved, or deleted

## apropos & whatis

- **apropos**
  - Locate commands by keyword lookup
  - Syntax: *apropos keyword*
  - *Example:*
    urmac % *apropos compiler*
  - Note: *apropos* is no different than *man –k*
- **whatis**
  - Displays a one line summary about a command
  - Syntax: whatis command
  - Example:
    urmac % *whatis vi*

## Commands - Basic directory

(a directory is a file with hard links to other files)

- mkdir - makes a directory (synonymous to folder)
- rmdir - removes a directory
- cd - changes your directory
- pwd - print the current directory
- mv - will move a directory, and copy the soft links
  - it just cuts and regrafts the directory to a new place in the tree
- cp - will copy a directory, but not the soft links

Remember a hard link is a link to an inode, which points to the file, so moves and copies just copy the link to the inode and all is well - the inode points to the file. But a soft link, links to another filename, so if you move the filename, the link is lost, although the file is still there.

## Commands – Less Basic

•less basic:

- finger – get some information on other users (like last login)
- which – locate where a command is in the user's path
- whereis - similar
- diff – find the difference between two (text) files

## Groups and Permissions - 1

Groups
In Unix, all users belong to at least one group.
Each person in one group has similar access to the filesystem.

There are 3 categories of user:-
- the **u**ser,
- the **g**roup to which the user belongs
- and **w**orld – everyone else on the system.

Permissions
- Files and directories all have associated "permissions" .
- Permissions tell the OS who can do what with your files and directories.
- The permissions are:
  - **r**ead, **w**rite, e**x**ecute

## Groups and Permissions - 2

To see the permissions on a file, do a 'ls –l'

```
 urmac:/urcwd urid$ ls –l
 -r--r--r--  1 urid  urgroup  17375 Apr 26 2000  rgb.txt
 -rw-r--r--  1 urid  urgroup  17375 Apr  5 02:57 set10.csv
 drwxr-xr--  1 urid  urgroup   1024 Jan 19 19:39 tests

mode     #links urid  urgroup  blocks last modified name
```

Changing Permissions
chmod [ugo]+[rxw] <filename>
e.g. chmod u+w rgb.txt ← gives me permission to change rgb.txt

Changing Ownership
chown <user>.<group> <filename>
e.g. chmod urid.iuns rgb.txt ← makes rgb.txt owned by iuns group

## Wildcards and Redirection

• Wildcards:
? – matches a single character
* – matches any number of characters
[xy] – matches either x or y (note: you can use other letters too!)

Redirection:
> – pipes standard output (printf, stdout, cout) to a file
>> - appends standard output to a file
< – pipes a file into standard input (scanf, stdin, cin)
<<END
- treats all text following as std input until END word encountered
| – connects the output of one command to the input of another
0, 1, 2 with < or > can be used to designate stdin, stdout, stderr
e.g.
stderr & stdout can both be redirected to a file by 1>file 2>file

## Groups and Permissions - 3

```
 -r--r--r--  1 urid  urgroup  17375 Apr 26 2000  rgb.txt
 -rw-r--r--  1 urid  urgroup  17375 Apr  5 02:57 set10.csv
 drwxr-xr--  1 urid  urgroup   1024 Jan 19 19:39 tests
```

user group other

What they do:
files: read – Allows you to read the file
write – Allows you to modify the file
execute – Allows you run the file as a script or binary program
(***!!!Common reason beginner's code won't run!!!!***)

directories:
read – lets you get a directory listing of files
write – lets you add or remove files from directory
execute – lets you access files in the directory

## Login Scripts (what happens at startup)

- .login, .bash_login, .mylogin, .profile, .bash_profile
  **One** of these scripts get run at startup…. If you use bash, you should check your .bash_profile so that you know what is getting run. (? reset automatically by central system here !?)
- .logout, .bash_logout
  **One** of these gets run when you logout or quit a shell
- .cshrc, .mycshrc, .bashrc
  One of these get run you start a new shell (as opposed to logging in. An example would be when you start a new xterm). .cshrc and .mycshrc are run if you use tcsh and .bashrc is run if you use bash.
- .xsession
  This gets run if you login via xdmcp. This is the only thing that gets run. It must be set as executable (chmod u+x) to work.

## Commands - Basic process

- ps – list the current processes in the shells
- kill – kill processes
  - (-9 if you want to kill them violently
    - I.e.– no hangups – or warnings)

## System State: uptime, top, nice

- Sometime you want to know what process is taking up the most cycles on a machine. The program that tells you this is "top"
- Another measurement utility for system load is "uptime." uptime tells you how long the system has been running and how many processes are concurrently screaming for attention. The last three numbers tell you how the average number of processes screaming for attention over the last 1 minute, 5 minutes and 10 minutes, respecetively.

```
4:49pm up 1:15, 19 users, load average: 0.10,
0.11, 0.09
```

- If you are going to run a long, cpu intensive process, it is often a good idea to "nice" it. This puts it at a slighly lower priority so that quick, interactive tasks (like checking e-mail) won't get starved. To do "nice" a process, just do "nice <command as you would>"

## Processes 1: suspend, background, foreground

- When you run a command, it starts a new "process"
- Each process has a unique number : the PID (Proccess ID)
- Unix is a multitasking operating system, so you can run multiple processes simultaneously
- After you start a process, it is usually in what is called the "foreground." That is, it takes over your shell.
- You can suspend processes in the foreground with Ctrl-Z
- The process is now frozen. You can pull it to the foreground again by typing "fg".
- Alternately, you can make it keep running, but put it in the background so can still use the same shell by typing "bg".
- You can also start a task in the background by putting a & at the end of the command.

## Environment Variables

What are environment variables? Think of them as parameters that get passed to your programs. All operating systems have them. They give information about the context within which a program runs. The most commonly used environment variable is the PATH variable which tells the shell where to look for programs.

To create an environment variable in bash, type:
export VARNAME=value

To create an environment variable in tcsh type:
setenv VARNAME value

Don't forget the export or the setenv, otherwise you create a "Shell variable" instead of an environment variable.

## Processes 2: ps, kill, kill -9

- You can list all the processes that have been run from the current shell with "ps"
- To list all the processes on the system, do "ps awux"

```
root    3723 0.0 0.1 3092 988 pts/20  S 16:21 0:00 -bash
urid    3724 0.0 0.1 1406 712 pts/17  R 16.35 0:00 -bash
urid    3725 0.0 0.1 2334 716 pts/17  R 16.36 0:00 ps awux
```

- If you want to end a process, you can do "kill <pid>" eg. kill 3724
- If that doesn't kill it, you can do the "super kill," "kill -9 <pid>" eg. kill –9 3724

If a process of yours freezes, you can login to the same machine again, do a "ps awux" and find the process number and then kill it.

## Common Environment Variables

- $HOME - home directory of current user
- $PATH - colon separated list of directories for commands
  e.g. to add current directory PATH = $PATH : .
- $PS1 - command prompt .. usually $ .. but can redefine!
- $PS2 - a secondary command prompt …
  - when prompting for more input ..usually >
- $IFS - Input Field Separator … to separate words
  (to cheat around filenames with spaces…
  … just redefine IFS omitting spaces,
  BUT CAN CAUSE OPPOSITE PROBLEMS!
  ..i.e. if you want to include some with spaces!)
- $0 - name of shell script
- $1 - name of first parameter… etc. $2, $3
- $# - number of parameters passed
- $$ - process ID of shellscript e.g. for temporary files