Shell (from Steve Bourne tutorial)

- The shell is a command programming language that provides an interface to the UNIX operating system.
- Its features include control-flow primitives, parameter passing, variables and string substitution.
- Constructs such as **while**, **if then else**, **case** and **for** are available.
- Two-way communication is possible between the shell and commands
  - String-valued parameters, typically file names or flags, may be passed to a command
  - A return code is set by commands that may be used to determine control-flow, and the standard output from a command may be used as shell input

Shell

- The shell can modify the environment in which commands run.
- Input and output can be redirected to files, and processes that communicate through 'pipes' can be invoked.
- Commands are found by searching directories in the file system in a sequence that can be defined by the user.
- Commands can be read either from the terminal or from a file, which allows command procedures to be stored for later use.

Shell Operation

- Reads its input
  - from the user's terminal
  - from a file
  - from a string supplied as an argument to the `-c` invocation option
- Breaks the input into words and operators, obeying the quoting rules. These tokens are separated by metacharacters.

  `|`, `&`, `;`, `(`, `)`, `<`, `>`

  Alias expansion is performed by this step

Shell Operation

- Parses the tokens into commands
- Performs the various shell expansions breaking the expanded tokens into lists of
  - filenames
  - commands
  - arguments
- Performs any necessary redirections
- Executes the command
- Optionally waits for the command to complete and collects its exit status

Shell

- Is the external layer of the Operating System that offers a user interface
- A Unix shell is
  - a command interpreter, which provides the user interface to the rich set of Unix utilities
  - a programming language, allowing these utilities to be combined
- It allows
  - Interactive dialogue
  - Batch processing through script files
- A Unix shell is not part of the kernel, but a user process

Shell Execution

- A shell process can be activated
  - Automatically during the login (according to a specific field in `/etc/passwd`)
  - Nested to another shell
    - Returns to the initial shell when exits
- To terminate a shell
  - `exit`
  - EOF character (`CNTR d`)
  - `logout`

Special Characters

- `/` path separator
- `?` a single character
- `*` a sequence of characters
- `~` login directory
- `~user` login directory of `user`
- `[]` a character among the list in brackets
- `{ }` a word among the comma separated list in brackets
- `'…'` does not expand regular expressions

Shells

- Several shells have been implemented, the most used are:
  - Bourne shell (`sh`): the original shell, often used for system programming
  - C-shell (`[t]csh`): the Berkeley shell, more C oriented, good for interactive use
  - Bourne again shell (`bash`)

Configuration Files

- When a shell is executed, it search in the user login directory its configuration files:
  - `.login` (`csh`, `tcsh`): commands executed at login
  - `.cshrc` (`csh`, `tcsh`):
    - commands executed whenever the shell is run
  - `.profile` (`sh`, `bash`): commands executed at login
- `csh` e `tcsh` use also `.logout` to perform some actions before terminating the session.

Shell Characteristics

- Completion
- Regular expressions
- I/O Redirection
- Pipeline
- History
- Aliasing
- Process management
- Scripting
- Variables

Completion

Expands the file names using **TAB** up to the next ambiguity

- File names having the execution right are searched in the directories listed in the **path** variable
- File names that do not have the execution right are searched in the working directory

Regular expressions

- The shell expands the regular expressions
- The regular expressions are replaced with the list of the file names that match the regular expression

Regular Expressions

```
$ ls
   file1
   file2
   rc.conf
   myconf.txt
```

```
$ ls -l file*
```

Shell

```
$ ls -l file1 file2
```


Regular Expressions

```
> ls
   file1
   file2
   rc.conf
   myconf.txt
```

```
$ ls -l *conf*
```

Shell

```
$ ls -l rc.conf  myconf.txt
```


Regular Expressions

```
$ ls
   file1
   file2
   rc.conf
   myconf.txt
```

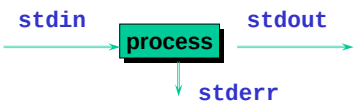
```
$ ls -l '*conf*'
```

Shell

```
$ ls -l  '*conf*'
```

I/O Redirection

Every process has three standard I/O streams open:

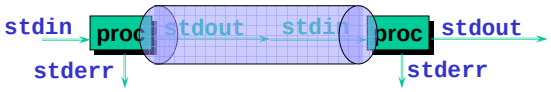


- Every channel can be redirected:
  - To a file
  - To another command through a pipe

I/O Redirection from/to file

- command < file — stdin **from** file
- command > file — stdout **to** file **(rewritten if it exists)**
- command >> file — stdout **appended to** file
- command <<HERE text HERE — stdin **from "here document"**
- command &> file — **(bash)** stderr+stdout **to** file
- command 2> file — **(bash)** stderr **to** file **(1 is** stdout**)**

Pipe



- A pipe connects an output stream to an input stream
  - The output produced by a process at its standard output is passed to the standard input of a process as it were typed on the keyboard.

I/O Redirection through a pipe

- command1 | command2
  - pipe between two commands
- Example:
  - ls -la | more

history Commands

- history — lists the history buffer of commands
- ↑↓ — shows the commands that have been previously executed
- !n — execute command number n in the history buffer
- !$ — last parameter of the previous command
- !string — last command beginning by string

Examples of use of history

```
25$ cc -g prog.c
26$ vi iop.c
27$ cc prog.c iop.c
28$ a.out letter
29$ rm !$          29$ rm letter
30$ !c             30$ cc prog.c iop.c
31$ !25            31$ cc -g prog.c
```

Aliasing

- It is possible to define new commands (alias) typically to speed up typing
- `alias`
  Lists the defined aliases
- `alias name=value`
  Defines an alias
- `unalias name`
  Remove an alias

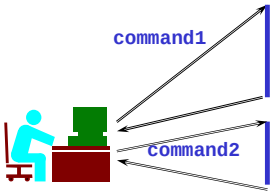
Example

- `alias dir="ls"`
- `alias tgz="tar czvf"`

Processes

- Linux is a multitasking Operating System
- It is possible to run several programs at the same time
- Shell commands can be executed
  - Sequentially (batch mode)
    - The user receives the system prompt only after the termination of the command
  - Concurrently
    - The user receives the system prompt immediately, and can issue a new command that runs in parallel with the previous one(s)

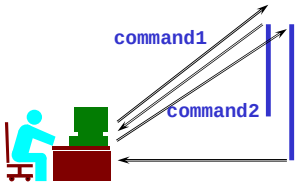
Sequential (Batch) Execution




```
$ command1
$ command2
```

The shell process waits the termination of process executing `command1`

Concurrent Execution

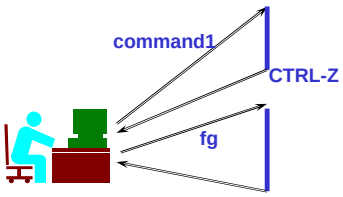



```
$ command1 &
$ command2
```

!!

The `&` send the process executing `command1` to the background. The shell process remains in the foreground

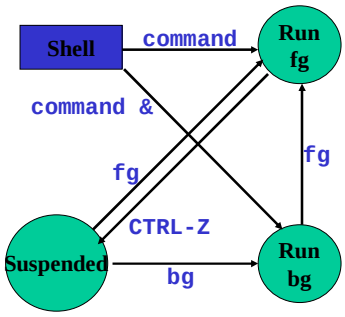
Interrupting a Process



Process state

- Executing in foreground
  - The three standard streams are connected to the session terminal :
  - 0) keyboard `stdin`
  - 1) video display `stdout`
  - 2) video display `stderr`
- Executing in background
  - The stdin stream is closed for the process executing in this state
- Suspended

Process state



Process Commands

- jobs lists the jobs in background
- bg %job-id move job %job-id to background
- fg %job-id move job %job-id to foreground

Processes

- A process has associated:
  - **pid** process id
  - **uid** user id of the user running the process
  - **stime** process starting time
  - **...**
- Command **ps** shows the user processes

Processes

- A process has associated:
  - **pid** process id
  - **uid** user id of the user running the process
  - **stime** process starting time
  - **...**
- Command **ps** shows the user processes

ps Command

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.0	1120	68	?	S	10:28	0:06	init [5]
root	2	0.0	0.0	0	0	?	SW	10:28	0:05	[kflushd]
root	3	0.0	0.0	0	0	?	SW	10:28	0:02	[kupdate]
root	5	0.0	0.0	0	0	?	SW	10:28	0:02	[kswapd]
bin	326	0.0	0.0	1208	0	?	SW	10:28	0:00	[portmap]
root	388	0.0	0.0	1492	0	?	SW	10:28	0:02	[klogd]
daemon	402	0.0	0.0	1144	104	?	S	10:28	0:00	/usr/sbin/atd
root	416	0.0	0.0	1328	112	?	S	10:28	0:00	crond
root	430	0.0	0.0	1144	124	?	S	10:28	0:00	inetd
root	444	0.0	0.0	1204	0	?	SW	10:28	0:00	[lpd]
laface	1309	0.3	0.3	2500	1552	pts/0	S	19:32	0:00	-csh
laface	1328	0.0	0.1	2324	692	pts/0	R	19:32	0:00	ps aux

Process Termination

- `kill -9 pid`
- `kill -9 %job-id`
- `killall –9 process-name`

Delayed Process Activation

- `at time filename`
  - Execute the process at the given time
- `at -l`
  - Shows the list of the scheduled processes in the cron queue
- `at -r [jobname]`
  - remove jobs from the cron queue

Commands (Script) Files

- A sequence of commands can be stored in a file, these commands will be executed “running” the script file.
- Indirect execution:
  - `source <scriptname> <args>`
  - `shell-name <scriptname> <args>`
- Direct execution, running the script
  - The file must have the execution right set
  - The first row of the file begins by `#!` Followed by the absolute filename of the shell that will interpret the commands. Ex: `#!/bin/bash`

Shell Script

```
#!/bin/bash

date
who
```


bash: Environment Variables

- Predefined variables. Ex.
  - `HOME`
- Custom variables
- `varname=value`
  - Set `varname` to `value`
- `set`
  - Shows all the environment variables, and their values
- `echo $varname` `!! Notice $`
  - Shows the value of variable `varname`

Environment Variables

- Some useful variables:
  - `home` login directory
  - `path` directories where the shell looks for executable files
  - `prompt` defines the style of the command prompt
  - `cwd` current working directory
  - `status` result of the last command
- Use `rehash` if the variable `path` is changed