7. The Shell

- Command Syntax and Job Control
- Classical UNIX Filters
- Shell Programming
Help Yourself

UNIX man pages

- Read man pages:
  http://www.linuxmanpages.com or http://linux.die.net/man
  
  ▶ Quick reference in French:
  http://www.blaess.fr/christophe/documents.php?pg=001
  
  ▶ For (our modified version of) Android in the labs:
  http://www.enseignement.polytechnique.fr/informatique/INF422/busybox.html

- Command-line usage

  ▶ $ man 1 command (UNIX command)
  ▶ $ man 2 system_call (primitive system calls)
  ▶ $ man 3 library_call (e.g., C library, system call front-end stubs)
  
  ▶ Warning: multiple entries with the same name may appear in different sections of the man pages
    → run $ man -k name if you are not sure
  
  ▶ The SEE ALSO section at the bottom of most man pages is an important way to navigate through this valuable source of precise/reference information
7. The Shell

- Command Syntax and Job Control
- Classical UNIX Filters
- Shell Programming
Shell Command Syntax and Examples

- **Command prompt**
  - User: $ or %
  - Root: #
  - Often prefixed by host name and/or current path for convenience

- **Space-separated command and arguments**

- **Argument convention:** – prefix for options
  - E.g., $ ls -l /usr/bin

- **--** for long option names in GNU tools
  - E.g., $ ls --version

- **List of common user commands:**
  - ls, cd, pwd, rm, mkdir, rmdir, chmod, cat, more, echo, ln, du, df, ps, kill, tar, ping, netstat, nc, exit

- **List of common root (administrator) commands:**
  - su, mount, fsck, mkfs, dd, passwd, uname, traceroute, ip, iptables, route, lsmod, modprobe, rmmod
More Shell Syntax

- Separate commands on a single line: `;`
- Variable expansion: `$VARIABLE`
  - E.g., `$HOME`, `$PATH`
- Setting a variable: `VARIABLE=string`
  - E.g., `HOME=/home/acothen`, `PATH=$HOME/android:$PATH`
- Export a variable to the process environment (inherited in child processes): `export VARIABLE`
  - Convention: exported variables use capital letters
More Shell Syntax

- Escape character to protect special characters: \n  - E.g., /home/my_name/long\ file\ name
  - Also used to extend a command to multiple lines

- Strings of uninterpreted characters
  - 'string'
  - "string" expands shell variables in string

- Most useful shell regular expressions
  - * matches any file name (without / or whitespace)
    - E.g., $ ls src/*.java
  - ? matches any letter except / and whitespace
## Interactive Shells

- **Automatic completion:**
  \(<\text{Tab}>\)

- **History:**
  \(<\text{Up}>, <\text{Down}>\)

- **End of input (a.k.a. end of file):**
  \(<\text{Ctrl-d}>\)

- **Clear terminal:**
  \(<\text{Ctrl-l}>\)

- **Interrupt the foreground process:**
  \(<\text{Ctrl-c}>\) (UNIX signal INTR = 2)

- **Quit the foreground process (failure):**
  \(<\text{Ctrl-\}}\) (UNIX signal QUIT = 3)
Job Control

- **Foreground** execution: `$ command arguments`
  - The command is executed in a child process and the shell waits for its completion

- **Background** execution: `$ command arguments &`
  - The command is executed in a child process but the shell does not wait for its completion
  - I.e., does not block further input from the shell, but cannot read any input from it either
Job Control

- Stop the foreground process:
  Ctrl-z
- Continue the last stopped process in the background:
  $ bg
- Continue the last stopped process in the foreground:
  $ fg or $ %%
- Continue process n in the stack of shell-controlled processes in the foreground:
  $ %n
- List shell-controlled processes:
  $ jobs
Redirect and Pipes

- Redirect standard output:
  $ command > file

- Redirect standard error:
  $ command 2> file

- Redirect standard input:
  $ command < file

- Redirect standard error to standard output:
  $ command > file 2>&1

- Chain standard output to standard input through a pipe (FIFO):
  $ command1 | command2
7. The Shell

- Command Syntax and Job Control
- Classical UNIX Filters
- Shell Programming
Pattern Matching in Texts: grep

Usage

$ grep \[options\] regexp \[file\] ...

Matches lines in a text file (or standard input) according to a regular expression pattern

Common options

- `\~v`: negate the regular expression
- `\~i`: case insensitive
Pattern Matching in Texts: grep

Basic Regular Expressions
- . matches any character
- * matches 0 or more occurrences of the previous
- \? matches 0 or 1 occurrence of the previous character/item character/item
- \+ matches 1 or more occurrences of the previous character/item
- [characters] matches those characters
- [^characters] matches everything but those characters
- [l₁–l₂] matches the range of letters from l₁ to l₂
- \| forms the union of two regular expression languages
- \(...\) forms a sub-expression
- ^ at the beginning of the pattern matches the beginning of the line
- $ at the end of the pattern matches the end of the line
- \ before a special character removes their special meaning, but makes ?, +, |, ( and ) special...
Filtering and Transforming Text: sed

Usage

- $ sed \([options]\) \([file]\)
  
  Automated edition of a text file (or standard input)
  
  Common options
  
  - `-e script`: add `script` to the edition commands
  - `-f script-file`: add the contents of the script file to the edition commands

Edition Cycle

1. Read a line into the *pattern space*
2. Remove the trailing `\n` (newline character)
3. Apply commands in sequence to the pattern space
4. Output the (edited) pattern space
5. Append a trailing `\n`
Filtering and Transforming Text: sed

Substitution Command

- \texttt{s/regexp/replacement/\[g\]}
  - Basic regular expressions (like \texttt{grep})
  - Substitutes the maximal string corresponding to the first match of regular expression in the pattern space
  - Matched text corresponding to the first 9 sub-expressions of the form \texttt{\(\cdots\)} can be substituted in the replacement text with \texttt{\1 to \9}
  - \texttt{&} substitutes the whole string matching the regular expression

- \texttt{g} flag: global substitution, beyond the first occurrence in the line

Other Commands: Full Automated Edition Language

- See \url{http://www.gnu.org/software/sed/manual/sed.html}
## Popular Scripting Languages

### Text Processing
- **awk**: pattern scanning and processing language, declarative (rule-based rather than imperative sequence of steps)
- **perl**: extension and unification of the shell, *sed* and *awk* in a single Swiss Army Knife language, with a large set of support libraries (called modules)
7. **The Shell**

- Command Syntax and Job Control
- Classical UNIX Filters
- Shell Programming
Shell Scripts

Typical Purposes
- Batch processing: periodic, scheduled, administrative tasks
- Programs dominated by I/O and text editing
- Bootstrap scripts and system configuration
- Composition of simple tools into complete programs
- Quick prototyping and run once programs

Beyond Shell Scripts
- Modern non-shell-based script languages: python, ruby, etc.
- Portable web-based languages: ECMA Script (a.k.a. JavaScript)
7. The Shell – Shell Programming

**Syntax**

**Script Structure**

- First line begins with `#!/absolute_path_to_the_shell`
  
  E.g., `#!/bin/sh`

- `#` introduces a comment

- A trailing `\` extends the current line to the following one

- A *list* is a sequence of one or more commands (or pipelines of commands) separated by one of the operators `;`, `&`, `&&` or `||`
7. The Shell – Shell Programming

Syntax

Example

#!/bin/sh

# Print the full name of the host system
uname -a
# Print the date twice within a one second interval (and a useless background sleep)
date; sleep 1 & sleep 1; date
Variables

- Set a variable: $ variable=value
  - Export a variable to the environment of child processes: $ export variable

- Variable expansion: ${variable}
  - Curly braces are optional, but required when the variable is immediately followed by some text
    - E.g., $ var=foo; echo ${var}bar echoes foobar
  - Default value: ${variable:=default_value}
    - Expands to default_value if the variable is unset (or empty string)
  - Many more special expansion syntaxes...

- Set a variable by reading a line from standard input: $ read variable
Special Variables

- $n$ expands to the $n$-th argument of the shell program (a.k.a. positional parameter)
  - $0$ is the shell script name itself
- $*$ expands to the space-separated concatenation of all arguments (except argument 0)
  - $@$ behaves identically, except when the expansion occurs within double quotes, where it expands into individual words: "$@" is identical to "$1" "$2" ...
- $# expands to the number of arguments
- $? expands to the exit code of the last command
- And several others...
Command Substitution

- `command` runs `command`, then replace the command substitution syntax with the standard output of the command
E.g., `ls -l 'which ls'`
Conditional Execution

**Boolean Condition**
- `test expression` or `[ expression ]`
- Semantics: exits with the `exit code` determined by `expression`
  - Warning: `true` translates into exit code `0`, and `false` to non-0!

**Conditional Statement**
- `if list; then list; [elif list; then list] ... [else list] fi`
## Conditional Expressions

### Primitive Expressions

- `-e file`: true if `file` exists
- `-f file`: true if `file` is a regular file
- `Similarly: -r, -w and -x for readability, writability and executability`
- `-n string`: true if `string` is not empty
- `String comparisons: string_1 op expression_2 where op can be = or !=`
- `string_1 != expression_2`: true if both strings are not equal
- `Integer comparisons: integer_1 op integer_2 where op can be -eq, -ne, -lt, -le, -gt, -ge`

### Compound Expressions

- `! expression`
- `expression_1 && expression_2`
- `expression_1 || expression_2`
- `( expression )`
Conditional Execution Example

Example

#!/bin/sh

# Initialize a variable to the default name of any system
default_name="localhost"

# If the host name is not the default one, print it
if [ "$HOSTNAME" != "$default_name" ]; then
    echo "$HOSTNAME"
fi
More Shell Constructs

- Executing a shell script within the current shell
  - `.` shell_script or `source shell_script`

- Loops
  - `for variable in words; do list; done`
  - Other constructs available (arithmetic expressions)

- Functions
- Built-in functions
- And others