

Basic Shell Commands

Marek Kozłowski



Faculty of Mathematics and Information Sciences
Warsaw University of Technology

Outline of the Lecture

- 1 Manual
- 2 Session Utilities
- 3 Gathering User and System Information
- 4 File Viewing and Editing
- 5 File Management
- 6 Dealing with Archives
- 7 File Finding

Outline of the lecture

- 1 Manual
- 2 Session Utilities
- 3 Gathering User and System Information
- 4 File Viewing and Editing
- 5 File Management
- 6 Dealing with Archives
- 7 File Finding

Manual (*man*)

- For help on any command, file name, system call, C function or general topic *man pages* are provided.

 We can display a man page on the *man* command:

```
Getting help on the man command
```

```
$ man man
```

- Man page navigation:
 - *q* – quit,
 - *<SPACE>* – next page,
 - *nG* – go to line number *n*,
 - *G* – go to the last line,
 - */pattern* – search forward for pattern; special characters in pattern need to be escaped,
 - *n* – next occurrence,
 - *N* – previous occurrence.

Man Page Structure

- Most man pages are organized following the same scheme:
 - NAME,
 - SYNOPSIS,
 - DESCRIPTION,
 - OPTIONS,
 - RETURN VALUE,
 - FILES (related files),
 - SEE ALSO (names with section numbers for man pages),
 - AUTHORS and BUG REPORTS.

Manual Sections

- The manual is organized into 8 sections numbered as follows (some System V systems can have 4, 5 and 7 swapped):
 - 1 User Commands,
 - 2 System Calls,
 - 3 C Library Functions,
 - 4 Devices and Special Files,
 - 5 File Formats and Conventions,
 - 6 Games et. Al.,
 - 7 Miscellanea,
 - 8 System Administration tools and Deamons.
- Supplemental POSIX sections can be defined. They share the same numbers followed by the letter **P** (for example: **1P** for POSIX-compliant commands).
- Names with numbers in parenthesis (for example: **kill(1)**, **kill(3P)**) refer to man pages in specific sections.

Selecting Manual Sections

- `man` with no options displays the first matching man page.
- We can specify the section number for `man` to seek.
- When in doubt we can request man pages on given name from all sections (the `-a` option). They are displayed in sequence.

 *Note that section numbers are not preceded with dash (-) opposite to other options:*

Specifying man sections

```
$ man kill # first match
$ man 1 kill # help on 'kill' command
$ man 2 kill # help on 'kill' system call
$ man -a kill # all man pages on 'kill'
```

Man Pages on Files and Functions

- Don't specify parenthesis for functions.
- Specify configuration or device files with no paths; use respective section numbers instead.
- Omit instance numbers.

Man pages on files and functions

```
$ man 3 printf # not: man printf()  
$ man 5 passwd # not: man /etc/passwd  
$ man 4 sd # not: man /dev/sda2
```


Outline of the lecture

- 1 Manual
- 2 Session Utilities
- 3 Gathering User and System Information
- 4 File Viewing and Editing
- 5 File Management
- 6 Dealing with Archives
- 7 File Finding

Secure SHell (SSH) Protocol

- Allows connecting a remote *tty* (or a *tty* emulator application) over a TCP/IP network.
- Provides a secure communication channel.
- Disconnecting or closing a terminal results in killing all processes within the current SSH session.
- Under Windows use the PuTTY client (<https://putty.org>):
 - works under 32-bit and 64-bit systems,
 - single exe file, no installation needed,
 - FLOSS license (open source).
- Under Unix use the `ssh` command.

ssh Command

Connecting to a remote host via SSH:

```
$ ssh [username@]hostname[.domainname]
```

- Username and domainname can be omitted if they are the same for client and server hosts.

- `screen` starts a new shell instance which is terminal independent. We can detach it from a terminal (and even log out) and attach to any other we are logged in if required.
- `screen` is especially useful when performing time-consuming tasks like: system upgrade, backup or complex computations.
- The most standard procedures include:
 - `screen` – start a session in screen,
 - `<Ctrl><A> <D>` – detach current session from terminal,
 - `screen -r` – re-attach a session running in screen,
 - `screen -rD` – re-attach a session running in screen; detach it first if necessary,
 - `exit` – end current session.
- If multiple `screen` instances are running, we are requested to select screen's ID when re-attaching (`screen -rD somenumber`).

- `script` makes a copy of everything printed on terminal and saves it into the `./typescript` file.
- `typescript` may contain non-ASCII characters (colors and other terminal control sequences).
- The `exit` command ends a `script`.
- `script` is commonly used inside sessions running in screen mode.

Outline of the lecture

- 1 Manual
- 2 Session Utilities
- 3 Gathering User and System Information**
- 4 File Viewing and Editing
- 5 File Management
- 6 Dealing with Archives
- 7 File Finding

Date and Time: *date*, *cal*

- `date` displays or sets date and time. There are syntax differences between implementations, consult your manual for details. For GNU `date` usage examples see *Shell Syntax Basics* (the slide: *Command Syntax Substitution*).

REMEMBER: For displaying the current time use the `date` command. The `time` command is designed for different purpose (measuring command execution time).

- `cal` displays a calendar: with no parameter the current month; `cal -y` – the whole year. Other options can vary.

System Information: *uname*

- `uname` displays system name (for example: *Linux*). Common querying options include:

- `-m` – hardware platform,
- `-n` – host name,
- `-p` – processor,
- `-r` – kernel,

`uname -a` prints all available information.

 Sample `uname -a` output for a Linux system:

uname information

```
$ uname -a
Linux alpha.mini.pw.edu.pl 2.6.36-gentoo-r5 #1 SMP
Thu Jan 27 12:47:37 CET 2011 x86_64 Intel(R) Xeon(R)
CPU L5420 @ 2.50GHz GenuineIntel GNU/Linux
```


Hardware Information (Linux)

- The following command may be available:

- `lscpu` ,
- `lspci` ,
- `lsusb` .

The `-v` options increases verbosity of the last two ones.

- Use some Linux LiveCD distro with those tools for examining a new hardware.

RAM Usage: *free*

- The `free` command displays information on total and free memory.
- Free memory can be used as buffers/cache for increasing system performance. It can be freed immediately shall the need arise.
- Some implementations support the `-h` option (human readable output – in kilobytes, megabytes, gigabytes, etc. rather than bytes).

User Identity: *id*

- The `id` displays the user and group names and IDs.
- Querying other users by name is possible with `id username`.
- The two utilities: `whoami` and `groups` which display current user name and user groups names are now obsoleted by `id -un` and `id -Gn username` respectively.
- A separate presentation is dedicated to user identities and groups.

User Information: *finger* (obsolete)

- *finger* is a user information lookup program.
- It is possible to search by user's system or real names.
- The mail information can be inaccurate for some mailbox formats. Other information (for example: a PGP key) may be included.

 *Information returned by *finger*:*

Getting user information

```
$ finger smith
Login: smithj                Name: John Smith
Directory: /home/smithj     Shell: /bin/bash
Last login: Fri Mar 4 08:00 (CET)
      on pts/0 from somehost.somedomain.com
No mail.
No plan.
```

Currently Logged In Users: *who*, *w*

- `who` displays who is on the system.
- `w` displays logged in users and the commands they're running.
- `write username [terminal]` sends a message to a user.
Specifying the terminal is necessary only if a user is logged on more than once.
- `mesg [y/n]` enables/disables incoming messages. With no parameter current status is displayed.

Outline of the lecture

- 1 Manual
- 2 Session Utilities
- 3 Gathering User and System Information
- 4 File Viewing and Editing**
- 5 File Management
- 6 Dealing with Archives
- 7 File Finding

- `echo` displays a line of text interpreted by the shell. It is commonly used for testing commands with special characters.
- `cat` displays concatenated files content on stdout (default: the screen). With `-n` it precedes each line with a line number.

File Viewing (1): *more*, *less*

- **more** displays files in paged mode (one screen a time). The `<SPACE>` key scrolls one page down. No backward scrolling is possible.
- **less** is the most advanced file viewer. It is used by default by **man** for displaying man pages, so the same navigation commands do apply. **less** is especially useful for examining large files (for example: log files).

File Viewing (2): *head*, *tail*

- **head** displays the first part of files (by default: 10 lines). The **-n** option allows specifying the number of lines.
- **tail** is similar to **head** except it prints the last lines. The **-f** option appends output when the file grows. It may be useful for log file tracing.

 A default log file is */var/log/messages*. Normal users are usually not allowed to read it:

Tracing a default *syslog-ng* log file

```
# tail -f /var/log/messages
```

Easy Editors

- For the very basic editing of ASCII files one or more of the following editors may be available:
 - **nano** (or **pico**) – a GNU editor default for most Linux systems. The list of available commands is displayed in the bottom panel. The ^ symbol denotes the <CTRL> key,
 - **joe** – simple editor similar to **nano**. The help panel is available too, all time the status bar informs:
Ctrl-K H for help,
 - **ee** (*Easy Editor*) – the basic editor for BSD systems.
- The **EDITOR** environment variable defines the default editor.

Outline of the lecture

- 1 Manual
- 2 Session Utilities
- 3 Gathering User and System Information
- 4 File Viewing and Editing
- 5 File Management**
- 6 Dealing with Archives
- 7 File Finding

Directory Navigation (1) – *pwd*, *ls*

- *pwd* (*Path to Working Directory*) displays working (current) directory.
- *ls* lists directory contents sorted by names. A file name instead of directory name can be given. If no name is specified it lists current directory. The most common options include:
 - *-a* – include entries which names start with '.' (hidden files),
 - *-A* – same as *-a* except . and .. directories,
 - *-l* – long listing (rights, owners, size, modification date etc.),
 - *-R* – recursive listing,
 - sorting options:
 - *-r* – reverse order,
 - *-S* – by size,
 - *-t* – by modification time,
 - *-X* – by extension.

Directory Navigation (2) – `cd`

- `cd` – changes working directory. Relative names are allowed. If no directory name is specified it changes to the directory stored in `$HOME`. We can change to the most recent directory by `cd -`.

✎ Let's assume that our home is `/home/smithj` and we just switched working directory to the `/etc`. The following commands are equivalent:

Changing working directory

```
$ cd /home/smithj
$ cd
$ cd $HOME
$ cd ~
$ cd ../home/smithj
$ cd -
```

Creating New Files and Directories: *touch*, *mkdir*

- `touch` updates file timestamps (*access* and *last modification* times). If the file doesn't exist a new empty one is created.
- `mkdir` creates directories. `mkdir -p` creates a multi-level structure if necessary.

 Note that `mkdir -p` doesn't return error on directory presence:

Creating a multi-level directory structure

```
$ mkdir sub1/sub2/sub3
    mkdir error: No such file or directory!
$ mkdir -p sub1/sub2/dub3
$ mkdir sub1/sub2/sub3
    mkdir error: File exists!
$ mkdir -p sub1/sub2/dub3
```

Copying, Moving and Renaming – *cp*, *mv*

- *cp* copies files, *mv* – moves/renames:

cp and *mv* syntax

```
$ cp source destination
```

```
$ mv source destination
```

- For copying directories with their contents the *-r* or *-R* (recursive) option must be present.
- The *destination* may indicate a new name or a new location.

 In the first example *somefile2* denotes a new name.
In the second one – *somedir* is a new location:

Copying a file

```
$ cp somefile somefile2
```

```
$ cp somefile somedir
```

Symbolic Links and Hard Links – *ln*

- Two kinds of file links do exist:
 - Two or more file system names that refer to the same file are called *hard links*.
 - A file reference by its name, similar to Windows *shortcut*, is called a *symbolic link* (or: *symlink*).
- Symlinks are not automatically updated on file name change.
- Hard links must exist within a single file system, while symlinks can refer to files in other file systems.
- Under normal circumstances only symlinks should be used.
- `ln` creates hard links; `ln -s` creates symlinks. The syntax is as follows:

ln syntax

```
$ ln [-s] target link-name(s)
```


Removing Files and Directories – *rm*

- `rm` removes files or directories.
- If there are hard-linked names then `rm` removes single name entries not the files they refer to.
- For removing directories the `-r` or `-R` (recursive) option must be present (obsoleted `rmdir` command can remove only empty directories).
- Other useful options include:
 - `-i` – prompt before deleting,
 - `-f` – never prompt, assume *yes*.

Synchronization – *rsync*

- **rsync** synchronizes destination directory to source directory. All files in the source directory are left untouched.
- **rsync** is able to synchronize directories between different hosts.

Typical *rsync* usage

```
$ rsync -avz --delete ~/sourcedir me@otherhost:/home/me/destdir
```

- Options explanation:
 - **-a** – preserve standard file attributes,
 - **-v** – be verbose,
 - **-z** – compress during the transfer,
 - **--delete** – delete extraneous files from destdir.
- The **-n** (or **--dry-run**) option starts **rsync** in *pretend* mode.

File Comparison – *diff*

- `diff` compares files line by line. `diff -y` (console or maximized terminal window is recommended) displays results side by side (in two columns). The symbols `<`, `>` and `|` denote respectively: the lines missing in the first file, the second one and the lines that differ.
- `diff` (with no options) output shows modifications that turn the first file into the second one. This output can be saved as a *patch* file for the `patch` command (see the next slide).

 *`file.patch` stores information on how `file.old` should be changed to be identical with `file.new`*

Generating a patch file

```
$ diff file.old file.new > file.patch
```

Applying *diff* Changes – *patch*

- `patch` allows applying changes as described in a *patch file*:

Applying a patch

```
$ patch file.old file.patch
```

Reversing applied changes

```
$ patch -R file.old file.patch
```

Outline of the lecture

- 1 Manual
- 2 Session Utilities
- 3 Gathering User and System Information
- 4 File Viewing and Editing
- 5 File Management
- 6 Dealing with Archives**
- 7 File Finding

Compression: *gzip* and *bzip2*

Compression

```
$ gzip somefile1  
$ bzip2 somefile2  
$ xz somefile3
```

- `gzip`, `bzip2` and `xz` compress files and add `.gz`, `.bz2` or `.xz` extensions.
- More than one file can be specified as an argument. In such case each file is compressed separately.

Decompression

```
$ gunzip somefile1.gz  
$ bunzip2 somefile2.bz2  
$ unxz somefile3.xz
```

- `gunzip`, `bunzip2` and `unxz` decompress files and remove extensions (`.gz`, `.bz2` or `xz`).

tar – Tape ARchiver

- `tar` creates a single file (a *tarball*) being a concatenation of files given as arguments:

Tarball creation

```
$ tar -cvf tarball.tar somefile1 somefile2 ...
```

Tarball extraction

```
$ tar -xvf tarball.tar
```

- The options:
 - `-c` – create,
 - `-x` – extract,
 - `-v` – verbose mode,
 - `-f` – specifies tarball name (usage: `-f tarball.tar`).
- Note that `tar`:
 - leaves source file(s) untouched,
 - doesn't automatically add the extension: `.tar` while creating a tarball.

Compressed Tarballs

- For multi-file archives we can first use `tar` then compress it with `gzip`, `bzip2` or `xz`:

Tarball compression with gzip

```
$ tar -cvf tarball.tar somefile1 somefile2 ...  
$ gzip tarball.tar
```

- Alternatively `-z` (for `gzip`), `-j` (for `bzip2`) or `-J` (for `xz`) options can be used:

Creating compressed tarball

```
$ tar -czvf tarball.tar.gz somefile1 somefile2 ...  
$ tar -cjvf tarball.tar.bz2 somefile1 somefile2 ...  
$ tar -cJvf tarball.tar.xz somefile1 somefile2 ...  
$ tar -xzvf tarball.tar.gz  
$ tar -xjvf tarball.tar.bz2  
$ tar -xJvf tarball.tar.xz
```


Compressed Tarballs Under Windows

- Most Windows archive utilities are able to deal with compressed tarballs.
- Windows utilities better deal with single extensions, so the following ones may be more safe:
 - `.tgz` - for gzipped tarballs,
 - `.tb2` - for bzipped tarballs,
 - `.txz` - for xzipped tarballs.

Outline of the lecture

- 1 Manual
- 2 Session Utilities
- 3 Gathering User and System Information
- 4 File Viewing and Editing
- 5 File Management
- 6 Dealing with Archives
- 7 File Finding**

File Finding – *find*

find syntax:

```
$ find [path] [tests] [actions]
```

- The parameters:
 - *path* – in which directory search for files (empty: the current one),
 - *tests* – search conditions; may be combined using logical operators: **!** , **-a** (*and*; default), **-o** (*or*) and parenthesis: **()** ,
 - *actions* – actions to perform on files that meet the tests (empty: print file names).

find – Tests and Actions

■ Common tests:

- `-name` – search by name (wildcards allowed)
- `-user` , `group` – by owner and group (numeric values allowed),
- `-type` – by type (`f` , `d` , `l` , `c` , `b` , ...),
- `-perm` – by permissions (symbolic or numerical),
- `-mtime` – by modification time (`+n` – more than `n` days, `-n` – less than `n` days, `n` – exactly `n` days).

■ Common actions:

- `-print` – print file names; the default one,
- `-delete` – delete files,
- `-exec command ;` – execute a command on files; for example:
`-exec rm {} \;` does the same as `-delete` (backslash protects semicolon to be not interpreted by the shell).

find – Examples

Search for relatively new configuration files.

```
$ find /etc -mtime -30
```

Move old backups to backup directory

```
$ find thisdir -mtime +365 -name "*.bak" -exec mv {} backdir/ \;
```

Delete some trash interactively

```
$ find thisdir -name ".*~" -exec rm -i {} \;
```