Processes & Jobs

PROCESSES	

chkconfig [<option> [<service> [<availability>]]]	**--add** <service> adds a service under chkconfig management, **--del** <service> removes a service, **--level** <runlevel> specifies the runlevel of the system, **--list** / **--list** <service> lists all services / specified service and their/its availability {off | on} on all runlevels
	`# chkconfig httpd on`
	(enables the service at runlevels 2, 3, 4 and 5)
	`# chkconfig --level 234 qemu off`
	(disables the service at runlevels 2, 3, and 4)
	`$ chkconfig --list | egrep -i "rexec|rlogin|rsh"`
	(lists the specified services and their availability at all runlevels)

systemctl <command> [<service ...>] (implemented from RHEL 7)	**enable** enables a service, **disable** disables a service, **is-enabled** shows availability of a service ("service unit") automatically after system startup, **mask** prevents a service from being started automatically or manually, **unmask** undoes the effect of "mask", **--user** manages user systemd files (services)
	`# systemctl enable httpd`
	(enables the service automatically after system startup)
	`# systemctl mask postfix.service`
	(prevents the service from being started automatically or manually)
	`$ systemctl list-unit-files -t service`
	(lists all services and their availability)

service [<service>] <command>	**start** starts a service, **stop** stops a service, **restart** restarts a service, **reload** reloads a configuration file of the service, **status** shows the status of the service
	`# service cups restart` | `# /etc/init.d/cups restart`
	(restarts the specified service)
	`# service --status-all`
	(shows the status of all services)

systemctl <command> [<service ...>] (implemented from RHEL 7)	**start** starts a service, **stop** stops a service, **restart** restarts a service, **reload** reloads a configuration file of the service, **status** shows the status of the service ("service unit"), **is-active** prints whether the service is active (running) or inactive, **--user** manages user systemd files (services)
	`# systemctl restart cups`
	(restarts the specified service)
	`# systemctl list-units -t service`
	(shows the status of installed services)
	`# systemctl list-units -t service --state=running`
	(shows running services)
	`# systemctl list-units -t service --all`
	(shows the status of all services)

PROCESSES	
systemctl list-jobs (implemented from RHEL 7)	lists all jobs that systemd is currently executing
<command> [<argument ...>]	executes an executable program (binary file) in the current shell; the program is a shell builtin command or a command found in $PATH or in the specified path `$ date`
bash | **sh** [<command | file> [<argument ...>]]	executes a shell, command or file (even without execute bits set) in a subshell; the file is searched in the current directory and in $PATH, **-r** executes a restricted shell, **-c** <string> reads commands from the string, **-s** reads commands from STDIN, **-x** executes the script in a debug mode `$ sudo bash -c "echo 'Welcome to my website' > /var/www/html/index.html"` (redirects the output of the command to the specified file without having to log in to the root user account)
source | **.** <file> [<argument ...>]	executes a file (even without execute bits set) in the current shell; the file is searched in the current directory and in $PATH `$ . ~/.bash_profile` (refreshes a user profile without the necessity to log off and log on)
<file> [<argument ...>]	executes an executable file in a subshell; the file is searched in $PATH or in the specified path `$ ./script`
command <command> [<argument ...>]	executes a shell builtin command or a command found in $PATH, ignoring any shell function named the same as the command name (the shell searches for a command name in this order: shell functions, shell builtin commands, executable files in $PATH)
builtin <command> [<argument ...>]	executes a shell builtin command, ignoring any shell function named the same as the command name (the shell searches for a command name in this order: shell functions, shell builtin commands, executable files in $PATH)

exec [<command> [<argument ...>]] [<redirection ...>]	replaces the shell with the specified command (no new process is created) or performs redirections
	`$ exec bash`
	(replaces the shell with bash)
	`$ exec > output.txt`
	(redirects all output to the "output.txt" file for the current shell process)

pstree [<PID | user>]	displays a tree structure of processes in alphabetical order, starting with the "init" (systemd) process or the specified PID or processes owned by the specified user, **-n** in numerical order (by PID), **-p** prints PID, **-u** prints usernames

PROCESSES	

ps	displays a static list of running processes of the logged-in user on the particular terminal (PID – the process identifier, TTY – the terminal, from which the process started (the "?" character means that there is no terminal associated with the process), TIME – the time the process has been processed by the processor, CMD – the command or process and its parameters), **-e** all processes of all users, **-f** a detailed output (shows also the UID – the owner of the process, PPID – the parent process identifier, C – the CPU usage, STIME – the time the process started), **-l** an entire output (shows also F – the process flag (0 – a regular process, 1 – a forked, but not executed process, 4 – a process running with root privileges, 5 – the combination of the previous values), S – the process status ("R" a running process, "S" a sleeping process (waiting for an event to complete), "D" a sleeping process not responding to signals (usually I/O), "T" a stopped (suspended) process, "Z" a zombie (defunct) process – a terminated process that still occupies a PID as the parent process is not aware of its exit status), C – the CPU usage in %, PRI – the priority of the process, NI – the priority value specified by the "nice" command, SZ – the total memory size in blocks occupied by the process, WCHAN – the address of the kernel function where the process waits for a particular event (a running process has "-")), **-p** <PID> a specified process, **-C** <program> processes of the specified program, **-U** <user | UID> processes of a specified user, **-G** <group | GID> processes of a specified group, **-t** <terminal> processes associated with a specified terminal, **-o** <format> according to the specified format, **--sort** <specification> sorts processes according to the specification, **-Z** prints the SELinux security context
	`$ ps -ef | grep sshd`
	(searches the running processes for the specified process)
	`$ ps -lt pts/1`
	(prints a detailed list of running processes on the specified terminal)
	`$ ps -p $$`
	(prints the current shell process)
	`$ ps -p 1 -o comm=`
	(prints the name of the process by its PID)
	`$ ps -eo pid,ppid,user,%mem,%cpu,comm --sort=-%cpu`
	(sorts processes by the CPU usage)
	`$ ps -eo pid,ppid,user,%mem,%cpu,comm --sort=-%mem`
	(sorts processes by the memory usage)

PROCESSES	
top	displays a list of processes in an interactive and dynamic way, including their PID, owner, priority or status, and overall CPU, memory and swap usage, **-u** <user> processes of a specified user, **-p** <PID> a specified process; interactive options: **f** displays a menu of items that can be added as an additional column to the command output (e.g., swap, UID or PPID), **M** sorts processes by memory usage, **P** sorts processes by processor usage (by default), **k** <PID> terminates the specified process, **q** quits the program
pidof <process_name>	prints the PID of a specified process `$ ps -p $(pidof sshd)` (searches the running processes for the specified process)
pgrep <pattern>	prints PIDs of processes matching the pattern, **-f** matches the full command line (not only the process name), including the executable path, arguments, and options, **-l** prints the process name as well, **-P** <PPID> prints processes of the parent process, **-c** prints the number of matching processes, **-i** ignores case distinctions, **-u** <user | UID> processes of a specified user `$ pgrep http` (prints PIDs of the httpd processes)

nice [[[**-n**] <priority>] <command>]	runs a process with modified priority ("nice" value), the "priority" parameter specifies the value to be subtracted from or added to the default value (0 by default), if omitted the value is automatically reduced by 10; the priority is specified in the range between -20 and 19, the lower the number, the higher the priority, negative values can only be set by root; with no argument the current inherited priority is printed
	`$ nice -n 15 rm -rf ~/tmp/*`
	(decreases the process priority by 15)
	`# nice -n -15 ls /etc | cpio -ov > /dev/rmt0`
	(increases the process priority by 15)

renice [**-n**] <priority> <identifier>	modifies the priority ("nice" value) of a running process, the value of the priority is specified in the absolute form in the range between -20 and 19; a regular user can only decrease the priority, [**-p**] <PID> specifies a process, **-u** <user> specifies processes of a particular user, **-g** <group> specifies processes of a particular group
	`# renice -n 5 987`
	(sets the priority of the specified process)
	`# renice -n -20 -u root`
	(increases the priority of all processes of the root user to the maximum)
	`# renice -n 19 -g users`
	(decreases the priority of all processes of the "users" group to the minimum)

Ctrl+**c**	terminates a running process in the foreground
Ctrl+**d**	terminates a running process that reads data from STDIN or exits the current shell

PROCESSES	

kill [<signal>] <PID ... | **%**job_number ...>	(**-15**) terminates a process or job matching the specified ID in a standard way (signal SIGTERM), **-9** terminates a process immediately (signal SIGKILL), **-1** terminates a process, for daemons it reloads their configuration file (signal SIGHUP), **-17** notifies the parent process on child processes events (signal SIGCHLD), **-19** suspends a process (signal SIGSTOP), **-18** starts a suspended process (signal SIGCONT), **-l** lists all signals, **-s** <signal> specifies a signal; the signal is specified by its number or name (with or without the "SIG" prefix and regardless of case)
	`# kill -9 1`|`# kill -SIGKILL 1`|`# kill -KILL 1`|`# kill -sigkill 1`|`# kill -kill 1`
	(terminates a specified process immediately)
	`$ kill %3`
	(terminates a specified job)
	`$ kill -s SIGCHLD 1376`
	(terminates a zombie process by sending the SIGCHLD signal to the parent process)

killall [<signal>] <process_name ...>	(**-15**) terminates all processes matching the specified name (especially useful for multiple running processes of one program) in a standard way (signal SIGTERM), **-9** terminates a process immediately (signal SIGKILL), **-19** suspends a process (signal SIGSTOP), **-18** starts a suspended process (signal SIGCONT), **-I** ignores case distinctions, **-l** lists all signals, **-s** <signal> specifies a signal, **-u** <user> terminates all processes of a specified user; the signal is specified by its number or name (with or without the "SIG" prefix and regardless of case) `# killall -9 sshd` (terminates specified processes immediately)
pkill [<signal>] <pattern>	(**-15**) terminates a process matching the specified pattern in a standard way (signal SIGTERM), **-9** terminates a process immediately (signal SIGKILL), **-19** suspends a process (signal SIGSTOP), **-18** starts a suspended process (signal SIGCONT), **-f** matches the full command line (not only the process name), including the executable path, arguments, and options, **-i** ignores case distinctions, **-u** <user | UID> processes of a specified user; the signal is specified by its number or name (with or without the "SIG" prefix and regardless of case)
fuser [<file ... | device ...>]	prints the processes using a specified file or device, **-m** prints the processes using the entire file system on which the file resides, **-u** prints the owner of the process, **-k** terminates a process, **-i** asks for confirmation, **-v** detailed output `$ fuser -kiv /dev/cdrom` (terminates processes using the specified device)
strace [<command>]	traces system calls and signals called and received by a process, **-e** <expression> traces a specified system call, **-o** <file> writes the output to a file, **-p** <PID> traces a running process, **-c** counts the time, calls and errors for each system call and reports a summary on program exit

PROCESSES	
ltrace [<command>]	traces shared library calls and signals called and received by a process, **-e** <expression> traces a specified library call, **-o** <file> writes the output to a file, **-p** <PID> traces a running process, **-c** counts the time and calls for each library call and reports a summary on program exit, **-S** additionally displays system calls
rpcinfo [<IP_address | hostname>]	**-p** prints RPC services (their identification number, version, protocol, port and name) running under the portmapper management on the local or specified computer; the list of all RPC programs is stored in */etc/rpc*
clustat	displays the status of the cluster
pcs status (implemented from RHEL 6)	displays the status of the cluster

JOBS	
jobs [**%**<job_number> ...]	prints a specified job (its number, "+" or "-" indicating the most recent or previous job, the status, and the command associated with the job) in the current shell, **-l** including PID, **-r** prints only running jobs, **-s** prints only stopped jobs, **-n** prints only the changes since the previous execution; with no argument all suspended jobs and jobs running in the background are printed
<command> **&**	executes a job in the background in a subshell
Ctrl+**z**	suspends a running job in the foreground
suspend	suspends the execution of the current shell until it receives a SIGCONT signal, **-f** suspends the login shell
bg [**%**<job_number> ...]	runs a specified suspended job in the background; with no argument the last suspended job is run
fg [**%**<job_number>]	runs a specified suspended job in the foreground or moves a specified job from the background to the foreground; with no argument the last suspended job is run or the last running background job is moved to the foreground
nohup <command>	executes a command in such a way that it is resistant to the SIGHUP signal (it runs even after the user has logged out); the output of the command is saved to the "nohup.out" file in the working directory `# nohup find /tmp -name core -print > core.txt &` (executes the specified command in the background, which runs even after the user logs out)

JOBS	

watch <command>	executes a command repeatedly at regular intervals of 2 seconds, **-n** <n> every *n* seconds, **-d** highlights the changes in the output, **Ctrl**+**c** terminates the program
	`$ watch -n 1 date`
	(displays the current time)
	`$ watch -d iostat`
	(displays changes to disks I/O loads and data write and read speeds)
	`$ watch "ps -eo %cpu,pid,user,args --sort=-%cpu | head -11"`
	(displays a dynamic list of the 10 processes that use the most processor)

wait [<PID | **%**job_number>]	waits for a specified process or job running in the background of the current shell to finish, then terminates; it returns the last process termination status, if the process doesn't exist, the return code is 127; with no argument it waits for all the current shell background processes to terminate and the return code is 0 `$ wait $! && echo "OK"` (waits for the last process running in the background to complete)
sleep <duration>	sets a time interval during which no activity is performed (especially useful when a certain delay is needed before executing the next command); the duration is specified by a number and a suffix **s** in seconds (by default), **m** in minutes, **h** in hours, **d** in days `$ sleep 10 && echo "10 seconds have passed."` (executes the following command after 10 seconds)

at <time> [<date>] [<additional_time_specification>]	executes a one-time job at the specified time; the commands to be run are read from STDIN or from a file, **-f** <file> reads commands from a specified file, **-c** <job_number> displays the contents of the scheduled job, **-d** <job_number> removes a scheduled job (equivalent to the "atrm" command), **-l** prints a list of scheduled jobs (equivalent to the "atq" command), **-m** sends an email to the user when the job has completed
	`$ at 20:00 25.06.2007 <-'` | `$ at 7am today + 3 weeks <-'`
	`at> mail root < offer.txt <-'`
	`at>` **Ctrl**+**d**
	(sends an email to the root user at the particular time with the contents of the specified file)
	`$ echo "date > date.txt" | at noon Sunday`
	(executes a command at the particular time, the output of which is written to the specified file)
	`$ at -mf list 20:00 25.06.2007`
	(executes a list of commands specified in the file at the particular time and sends an email to the user when the job has completed)

atq | **at -l**	prints a list of scheduled jobs (a job number, time of execution, a user who scheduled the job)
atrm | **at -d** <job_number ...>	removes a scheduled job

JOBS	
lpr [<file>]	prints a file; if no file is specified, reads from STDIN, **-P** <printer> specifies a printer, **-#** <n> specifies the number of copies
lpq	displays jobs in the print queue of the default printer, **-P** <printer> specifies a printer, **-a** jobs on all printers, **-l** detailed output
lprm [<job_number> | **-**]	removes a specified job from the print queue or the entire queue, **-P** <printer> specifies a printer; with no argument the current job on the default printer is removed
crontab [<file>]	configures an existing file to periodically execute the specified jobs via the cron daemon, **-e** creates or edits a crontab file, **-l** prints the contents of the crontab file, **-r** removes a regular crontab file, **-u** <user> sets up a crontab file of the specified user; the jobs are specified each on a separate line in the file, which in the case of user files consists of 6 fields separated by spaces – 5 for time specification (in the order of minute, hour, day of the month, month, and day of the week (Sun–Sat = 0–6), "*" matches all values, "*/"<n> every *n*th time interval, "," separates values of the same field, "-" indicates a range of values) and 1 for the command itself (the command must be specified using the absolute path, except for shell builtin commands and keywords, if it has any output, it is sent to the user by email); an empty line and a line starting with a "#" character are ignored; system files include */etc/crontab* and files located in */etc/cron.d*, which contain 1 extra field between the time specification and the command – the user under which the specified job should be run; user files are located in */var/spool/cron/<user>* and if they are edited manually (not with the "crontab -e" command), the changes are not atomatically loaded by the daemon; authorized users are specified in */etc/cron.allow*, unauthorized users in */etc/cron.deny*, if both the files exist, */etc/cron.deny* is ignored *30 8 * * * /usr/bin/df | /bin/mail admin* (executes a job every day at 8:30) **/30 8-16 * * 1-5 /usr/bin/who >> /tmp/users.out* (executes a job every 30 minutes in working hours) *0 22 * * 5 /usr/local/scripts/backup.sh* (executes a script every Friday at 22:00) *0 0 1 * * for file in $(/bin/find /web/logs/*); do > $file; done* (executes a job at the beginning of each month)