Linux Quick Reference Sheet

Contents

Disclaimer: This reference sheet is not meant to be comprehensive. It gives a quick review of the commands we have covered in lecture and how we used them.

Notation

Each section header is the command and its corresponding usage. Parameters that are optional are in square brackets (e.g. `[OPTION]`). If multiple parameters may be specified, the parameter is proceeded by `...` (e.g. `FILE...`).

In the example tables for each command, some commands may contain the \ character. This is common notation in commands to indicate the arguments continue on the next line. When typing the actual command, you may exclude the \ character and type the command on a single line.

1 Basics of Linux

1.1 `ls [OPTION]... [FILE]...`

Displays the file listing in each of `[FILE]...`

Command	**Behavior**

`ls`
`ls .`
	Display the contents of the current directory.

`ls DirA`	Display the contents of the `DirA` directory.
`ls -l`	Display more information about the contents of the current directory (file permissions, last modified, ...).

`ls -l -h`
`ls -lh`
	Same as `ls -l`, but shows the file sizes in human readable formats (e.g. 4K instead of 4096).

`ls -l -t`
`ls -lt`
	Same as `ls -l`, but sorts the listing by decreasing date modified .

1.2 `man page`

Gives a detailed description of the command specified by `page`. The page is scrollable using the arrow keys. To escape, press `q`. To search, press `/` and then start typing the regular expression search query. To find the next instance of the search, press `n`, and to find the previous instance of the search, press `N`.

Command	**Behavior**
`man ls`	Open the manual page for `ls`.

1.3 `pwd`

Shows the full path of the current working directory (CWD).

Command	**Behavior**
`pwd`	Display the CWD.

1.4 `cd [DIRECTORY]`

Changes the current working directory (CWD) to `DIRECTORY`. `DIRECTORY` can be an absolute path (e.g. `/home/users/chirag/DirA`) or a relative path (e.g. `DirA`) and may contain any combination of the following symbols:

- `.` (current directory)
- `..` (parent directory)
- `~` (home directory)

Command	**Behavior**

`cd`
`cd ~`
	Change the CWD to the home directory.

`cd DirA`	Change the CWD to the `DirA` directory in the CWD.
`cd ..`	Change the CWD to the parent directory.
`cd .`	Change the CWD to the current directory (no noticeable change)
`cd ../DirA`	Change the CWD to the `DirA` directory in the parent of the CWD.
`cd ~/DirA/DirB`	Change the CWD to `DirB`, which is in the `DirA` directory, which is in the home directory.

1.5 `mkdir [OPTIONS]... DIRECTORY...`

Creates new directories for each of `DIRECTORY`.

Command	**Behavior**
`mkdir DirA`	Create a new directory `DirA` in the CWD.
`mkdir ~/DirA/DirB`	Create the directory `DirB` in `DirA`, which is in the home directory.
`mkdir -p DirA/DirB`	Create the directory `DirB` in `DirA`, which may or may not exist. If it doesn't exist already, it will be created.

1.6 `rm [OPTION]... FILE...`

Deletes each of `FILE`. **Note that `rm` permanently deletes the file(s).**

Command	**Behavior**
`rm FileA`	Delete `FileA`.
`rm -r DirA`	Delete the directory `DirA`.
`rm -i FileA`	Delete `FileA`, but asks for confirmation before deleting.

1.7 `cp [OPTION]... SOURCE DEST` or `cp [OPTION]... SOURCES... DIRECTORY`

Copies `SOURCE` into `DEST`. If multiple sources are provided, the destination must be a directory.

Command	**Behavior**
`cp FileA FileB`	Copy `FileA` into `FileB`.
`cp FileA FileB DirA`	Copy `FileA` and `FileB` into directory `DirA`.
`cp -r DirA DirB`	Copy `DirA` to `DirB`. Create `DirB` if it doesn't exist, otherwise copies `DirA` into `DirB`.

1.8 `mv [OPTION]... SOURCE DEST` or `mv [OPTION]... SOURCES... DIRECTORY`

Moves `SOURCE` into `DEST`. If multiple sources are provided, the destination must be a directory.

Command	**Behavior**
`mv FileA FileB`	Rename `FileA` into `FileB`.
`mv FileA FileB DirA`	Move `FileA` and `FileB` into directory `DirA`.
`mv DirA DirB`	Move `DirA` to `DirB`. Rename `DirA` to `DirB` if `DirB` doesn't exist, otherwise move `DirA` into `DirB`.

2 File I/O and Processing

2.1 `echo [STRING]...`

Outputs each of `STRING` separated by new lines.

Command	Behavior
`echo 'Hello'`	Output 'Hello'.

2.2 `cat [OPTION]... [FILE]...`

Outputs each of `FILE`. If no file is specified, uses `stdin`.

Command	Behavior
`cat FileA`	Outputs the content of `FileA`.

2.3 `less [OPTION]... [FILE]...`

Outputs each of `FILE`. If no file is specified, uses `stdin`. The page is scrollable using the arrow keys. To escape, press `q`. To search, press `/` and then start typing the regular expression search query. To find the next instance of the search, press `n`, and to find the previous instance of the search, press `N`.

Command	Behavior
`less FileA`	Output the contents of `FileA` in a scrollable view.

2.4 `head/tail [OPTION]... [FILE]...`

Outputs either the beginning (`head`) or end (`tail`) of each of `FILE`. If no file is specified, uses `stdin`.

Command	Behavior
`head FileA`	Output the first 10 lines of `FileA`.
`tail -n 5 FileA`	Output the last 5 lines of `FileA`.

2.5 `find [PATH]... [EXPRESSION]`

Searches recursively in each of `PATH` with the constraints specified by `EXPRESSION`.

Command	Behavior

`find`
`find .`
	Output all the files and directories recursively in the CWD.

`find -type f`	Output all the files recursively in the CWD.
`find -type f -name '*File*'`	Output all the files recursively in the CWD that have the word `File` somewhere in their names.

`find -name '*.cpp' \`
`  -exec grep main {} +`
	Run `grep` on all `cpp` files in CWD (recursive).

2.6 `tee [OPTION]... [FILE]...`

Simultaneously outputs `stdin` and copies it to each of `FILE`. It makes the most sense to use this command at the end of a series of pipes (e.g. `cat FileA | tee FileB` will output `FileA` and copy it to `FileB`).

Command	**Behavior**
`tee FileA`	Output `stdin` and copies it to `FileA`.

2.7 `tr [OPTION]... SET1 [SET2]`

Converts, squeezes, and/or deletes characters from `stdin` and outputs the result. Sets are strings of characters, where each character is treated independently. For example, 'hi' is not treating as the word 'hi', but rather as the individual characters 'h' and 'i'.

Command	**Behavior**
`tr -s ' '`	Squeeze multiple consecutive spaces into a single space.
`tr -d ' '`	Delete all spaces.
`tr 'ab' 'cd'`	Replace all 'a's with 'c's and all 'b's with 'd's.

2.8 `cut OPTION... [FILE]...`

Outputs parts of a line (e.g. columns, characters, etc.) for each line in each of `FILE`. If no file is specified, uses `stdin`. Ranges can be specified as `start-end`, where `start` and `end` are inclusive. Some valid ranges include:

- `1` (only column 1)
- `1-2` (columns 1 and 2)
- `-3` (all columns up to column 3)
- `4-` (all columns from column 4 to the end)

It may be useful to pipe a `tr` command into `cut` (e.g. `cat FileA | tr -s ' ' | cut -d' ' -f1-2`).

Command	**Behavior**
`cut -f1-2`	Output columns 1 and 2, where columns are delimited by a single `TAB` character.
`cut -d' ' -f4`	Output column 4, where columns are delimited by a single `SPACE` character.
`cut -c-15`	Output the first 15 characters of each line.

2.9 `grep [OPTION]... PATTERN [FILE]...`

Searches for the regular expression `PATTERN` in each of `FILE`. If no file is specified, uses `stdin`.

See 4.1 on page 12 for information on regular expressions.

Command	**Behavior**
`grep 'Hello' FileA`	Output all lines in `FileA` that contain the word ’Hello’.
`grep '[A-Za-z]\+' FileA`	Output all lines in `FileA` that contain at least one letter.
`grep '.*' FileA`	Output all lines in `FileA` (since all lines match `.*`).
`grep '[0-9]\{2\}/[0-9]\{2\}'`	Output all lines in `stdin` that contain a date of the format `mm/dd`. Note that this doesn’t check for valid dates.
`grep '#include' *.cpp`	Output all the `#includes` in the `cpp` files in the current working directory.
`grep -r '#include' .`	Output all the `#includes` in all the files in the current working directory and all of the successive nested directories (recursive).

2.10 `sed [OPTION]... SCRIPT [FILE]...`

Executes the editing script `SCRIPT` on each of `FILE` and outputs the result. If no file is specified, uses `stdin`. `sed` scripts are composed of individual commands, which come in formats such as `s/regex/replacement/modifiers` or `/regex/d`.

One example of a command is search and replace, denoted by `s`. The search and replace command matches each line with `regex` and substitutes each match with `replacement`. By default, the substitution happens once per line, but the `g` modifier can be used to substitute all matches on a line. A search and replace may look like `s/[0-9]\+/12345/g`, which replaces all decimal numbers with 12345.

Another example of a command is delete, denoted by `d`. The delete command will remove entire lines if any part of the line matches `regex`. A delete operation may look like `/^[A-Z]\+$/d`, which will delete any line that only contains upper case characters.

See 4.1 on page 12 for information on regular expressions and 4.2 on page 12 for information on `sed`.

Command	**Behavior**
`sed 's/Hello/Hi/g' FileA`	Output all lines in `FileA`, but replaces all instances of ’Hello’ with ’Hi’.
`sed '/[A-Z][a-z]\+/d' FileA`	Output all lines in `FileA` other than those that contain a word that begins with a capital letter.
`sed 's/[A-Z][a-z]\+/DELETE/; \` `/DELETE/d' FileA`	Same as above, but uses multiple commands (the above is more efficient).
`sed -n '/12345/p' FileA`	Output all lines in `FileA` that contain the number 12345 (behaves the same way as `grep`).

2.11 `awk [OPTION]... PROGRAM [FILE]...`

Executes the editing program `PROGRAM` on each of `FILE` and outputs the result. If no file is specified, uses `stdin`. `awk` programs are composed of individual actions, which come in the format `pattern { action }`. `pattern` may be a regular expression or some other built-in `awk` patterns (e.g. `BEGIN`). `action` is a C-like sequence of statements.

`awk` comes with some built in variables that can be used in the actions. One example is that `awk` automatically applies a delimeter (default delimieter is a space) and generates columns. These columns can be accessed using `$n`, where `n` is the column number. For example, `awk -F',' '/.*/ { print $2; }' FileA` will output the second column of `FileA`, where columns are delimited by commas (equivalent to `cut -d',' -f2 FileA`). `$0` is a special variable that is the entire line instead of a single column.

Additionally, since actions consist of a sequence of C-like statements, you can define your own variables. The following will output all lines of `FileA` except for the first, because `start` is initially equal to `1`: `awk 'BEGIN { start=1; }; /.*/ { if(start==0) { print $0; } else { start=0; } }' FileA`. `BEGIN` is a special pattern for which the action will be executed exactly one time before any lines are processed. It is useful for initializing variables, as shown in the example.

See 4.1 on page 12 for information on regular expressions and 4.3 on page 12 for information on `awk`.

Command	**Behavior**
`awk '/.*/ { printf "%s %s\n", \ $1, $2; }' FileA`	Output the 1st and 2nd columns of `FileA`.
`awk '/[0-9]+/ { printf "%s \ %s\n", $1, $2; }' FileA`	Output the 1st and 2nd columns of `FileA`, if the line contains a decimal number.
`awk '/.*/ { temp=$1; $1=$2; \ $1=temp; print $0; }' FileA`	Output `FileA`, but swaps the 1st and 2nd columns.

3 Working Remotely

3.1 `ssh [OPTION]... [USER@]HOSTNAME`

Logs into the machine specified by `HOSTNAME` as `USER` and opens a shell.

Command	Behavior
`ssh \` `chirag@127.0.0.1`	Log into `localhost` as `chirag`.
`ssh -p2020 \` `chirag@127.0.0.1`	Log into `localhost` as `chirag` on port 2020. The SSH server must be running on port 2020.

3.2 `scp [OPTION]... [[USER@]HOSTNAME1:]FILE1... [[USER@]HOSTNAME2:]FILE2`

Securely copies each of `FILE1` from `HOSTNAME1` to `HOSTNAME2` over SSH. If a hostname is not given, uses `localhost` (the local machine). The semantics are similar to the `cp` command. Note that the direction of the transfer is specified by the order of the parameters (always `FILE1...` to `FILE2`), and `scp` should be run on the local machine unless using reverse tunneling.

Command	Behavior
`scp FileA \` `chirag@mario.ece.utexas.edu:~`	Copy `FileA` from the local machine to the home directory of `mario.ece.utexas.edu`.
`scp \` `chirag@mario.ece.utexas.edu:~/FileA \` `~/Downloads`	Copy `FileA` from the home directory of `mario.ece.utexas.edu` to the `~/Downloads` directory of the local machine.
`scp -P2020 FileA \` `chirag@mario.ece.utexas.edu:~`	Copy `FileA` from the local machine to the home directory of `mario.ece.utexas.edu` using port 2020.
`scp -r DirA \` `chirag@mario.ece.utexas.edu:~`	Copy the entire contents of `DirA` from the local machine to the home directory of `mario.ece.utexas.edu`.

3.3 `vim [OPTION]... [FILE]...`

A fully terminal-based text editor. Operations are performed from one of the following modes:

- Normal mode (start off in this mode): used to type commands, usually through one or more keystrokes.
- Insert mode: used to type like in a GUI-based text editor.
- Replace mode: used to overwrite characters while typing, similar to pressing the insert key and then typing in a GUI-based text editor.
- Visual mode: used to make selections.
- Ex mode: used to enter commands that do not have keystrokes associated with them, which would normally be used in normal mode.

When in normal mode, you can type commands and motions. Commands can several things, including switching to insert mode, deleting text, or copying and pasting. Some commands can be combined with motions. Any action that moves the cursor is considered a motion. This includes moving up or down, moving to the next word, or moving up a page. For example, to delete a word, you can type `dw` from normal mode. To delete 3 words, you can type `d3w` from normal mode. Below are some common motions and commands.

Motion	**Behavior**	**Motion**	**Behavior**
`h`	Move left	`l`	Move right
`j`	Move down	`k`	Move up
`w`	Move to next word	`b`	Move to previous word
Command	**Behavior**	**Command**	**Behavior**
`i`	Enter insert mode	`c`	Delete and enter insert mode*
`d`	Delete*	`x`	Delete a single character
`R`	Enter replace mode	`r`	Replace a single character
`y`	Yank/copy*	`p`	Paste

*Requires motion

See 4.4 on the next page for information on `vim`.

4 Useful References

4.1 Regular Expressions

- Tutorial
- Debugger

4.2 `sed`

- Tutorial

4.3 `awk`

- Tutorial

4.4 `vim`

- The `vimtutor` command
- Vim Adventures (tutorial game)
- Quick reference sheet