MRS lab ROS platform Cheat Sheet

by Tomas Baca @ Multi-robot Systems (MRS), v1.2.0

Ubuntu terminal - GNU/Linux basics

Hitting ⟨Tab⟩ autocompletes commands, filenames, etc.

New terminal	`Ctrl+Alt+t`
Need help	append `--help` after command
Need more help!	`$ man [command]`
Change directory	`$ cd [path]`
Path symbolic links	`.` – current directory `..` – previous directory `~` – home directory (also `$HOME`) `/` – root directory
create a file	`$ touch [path]`
remove a file	`$ rm [path]`
move (also rename) a file	`$ mv [from] [to]`
copy a file	`$ cp [from] [to]`
print a file	`$ cat [path]`
edit a file	`$ vim [path]`, `$ nano [path]`
set a variable	`$ VARIABLE="dog", VARIABLE=3.0`
print a variable	`$ echo "the content is: $VARIABLE"`
run a script or executable	`$ ./script.sh, ./program`
output redirection	`>` – to a file (rewrite) `>>` – to a file (append) `|` – pipe to another command
redirect to /dev/null	`> /dev/null 2>&1`

Would You Like to Know More? `https://ctu-mrs.github.io/docs/prerequisites/linux`

TMUX - Terminal multiplexer

Run tmux	`$ tmux`
List all sessions	`$ tmux ls`
Attach to a session	`$ tmux a -t [session name]`
New window (tab)	`Ctrl+t`
New horizontal split	`Ctrl+9`
New vertical split	`Ctrl+0`
Moving through windows (tabs)	`Shift+→`, `Shift+←`
Moving through panes (splits)	`Alt+→`, `Alt+←`, `Alt+↑`, `Alt+↓`
prefix	`Ctrl+a`
Killing window	**prefix** `x`, `$ exit`, `$ :q`
Killing session	**prefix** `k`
Detach from session	**prefix** `d`
Enter vim mode (scrolling, copying)	`F2`, **prefix** `[`

Would You Like to Know More? `https://github.com/klaxalk/linux-setup/wiki/tmux`

Vim – a modern modular text processor

Vim is not a joke. Although you might not know how to exit it (yet), it is a very powerful tool. Our vim is filled with features, including code snippets, code completion (ROS aware), code formatting, syntax highlighting and tmux integration. Its control is completely mouse-less and it is fully usable over ssh, which makes it great for remote editing on a drone. Moreover, its modal editing paradigm is very intuitive. Lastly, when you learn how to control vim, you also learn to control other tools such as *Linux manual pages*, *ranger*, *less* and much more. Even gmail uses vim-like controls natively. Run `$ vimtutor` to start learning vim using an interactive "file tutorial". Here are some simple commands:

switch to insert mode	`i`	jump a word/Word forwards	`w/W`
return to normal mode	`ESC`	jump a word/Word backwards	`b/B`
cut a line to clipboard	`dd`	change current word/Word	`ciw/ciW`
paste a clipboard	`p`	delete 3 lines down	`3dj`
open a command line	`:`	substitute *dog* for *cat*	`:%s/dog/cat/g`
save	`:w`	move cursor left/down/up/right	h/j/k/l
quit	`:q`	delete every line containing *dog*	`:%g/dog/normal dd`

Would You Like to Know More? `https://www.tutorialspoint.com/vim/`

Git version control system

Git is a distributed version control system. Repositories are equal, some are just used as a "server" (called **remote**). Git uses branches to isolate ongoing work on the same project. Branches can be merged to combine the work back into a single piece. Changes in the files should be **commited**. Only commit "runnable" code.

Cloning a repository	over ssh `$ git clone git@github.com:ctu-mrs/mrs_uav_core.git` over https `$ git clone https://github.com/klaxalk/linux-setup`
Update origin state	`$ git fetch`
Update current branch from remote	`$ git pull`
Update current branch to remote	`$ git push`
Commit "patch" – interactive	`$ git commit -p`
Add files for commit	`$ git add [file]`
Commit changes	`$ git commit -m "commit message"`
Checkout a branch	`$ git checkout [branch name]`
Create a branch	`$ git checkout -b [branch name]`
unstage the file	`$ git reset [file name]`
undo all uncommited changes	`$ git reset --hard`
remove all new unstaged files	`$ git clean -fd`
Merge a branch	`$ git merge [branch name]`
Rebase on a branch	`$ git rebase [branch name]`
refactor branch history	`$ git filter-branch [lot of args]`
show status	`$ git status`
show log	`$ git log`
show better log	`$ glog` – is an alias for `$ git log` with more arguments
show super forest log	`$ flog` – uses `/.scripts/git-forest.sh`

Would You Like to Know More? `https://try.github.io/`

~/.bashrc – Bash configuration

When a new terminal is opened and an instance of bash is launched, the `~/.bashrc` file is *sourced* (executed while its leftover variables, functions and aliases stay in the context). We use `~/.bashrc` heavily for setting context for ROS and our development environment. `~/.bashrc` sources ROS setup scripts, which are also generated by each workspace. If you change this file, source it (or open a new terminal) to activate the changes: `$ source ~/.bashrc` or just `$ sb`. Here is an example of what should not be missing in the bottom of a healthy `~/.bashrc` file:

```
source /opt/ros/lyrical/setup.bash

source ~/workspace/install/setup.bash
# source ~/other_workspace/install/setup.sh

# export ROS_WORKSPACE="$HOME/mrs_workspace"
export ROS_WORKSPACES="~/mrs_workspace ~/workspace"

export GIT_PATH=$HOME/git

export RUN_TMUX=true

# VARIABLES TO CONFIGURE THE MRS ROS PIPELINE
export UAV_NAME="uav1"
# ...
export MRS_STATUS="readme"

source $GIT_PATH/mrs_uav_development/shell_additions/shell_additions.sh

source $GIT_PATH/linux-setup/appconfig/bash/dotbashrc
```

ROS in Linux terminal

Please, visit `http://wiki.ros.org/ROS/Tutorials` before starting work on a bigger project.
Use ⟨Tab⟩ **to complete commands, topic names, message types and pre-fill message contents.**

Getting help	append `--help` after any following command
Listing all ROS nodes	`$ ros2 node list`
Listing all ROS topics	`$ ros2 topic list`
Listing all ROS services	`$ ros2 service list`
Listing all ROS params	`$ ros2 param list`
Running a ROS binary	`$ ros2 run package_name binary_name`
Running a launch file	`$ ros2 launch package_name file.launch.py`
Showing a node info	`$ ros2 node info /node/path`
Showing a topic info	`$ ros2 topic info /topic/path`
Showing a service info	`$ ros2 service info /service/path`
Showing a topic type	`$ ros2 topic type /topic/path`
Showing a service type	`$ ros2 service type /topic/path`
Showing all interfaces	`$ ros2 interface list`
Showing a specific interface	`$ ros2 interface show [type]`
Showing topic messages	`$ ros2 topic echo /topic/path`
Showing a param value	`$ ros2 param get /node/path /param/path`
Calling a service	`$ ros2 service call /service/path [args]`
Publishing on a topic	`$ ros2 topic pub /topic/path [type]`
Setting a param value	`$ ros2 param set /node/path /param/path [value]`

Would You Like to Know More? `https://docs.ros.org/en/lyrical/Tutorials/Beginner-CLI-Tools.html`

ROS workspace structure

MRS lab main workspace

path contains	`~/mrs_workspace/src/` – `mrs_uav_core` – core MRS repository – `mrs_uav_modules` – modules MRS repository

MRS lab student workspace

path contains	`~/workspace/src/` – `example_waypoint_flier` – general example – `mrs_computer_vision_examples` – templates

General ROS package structure

`build`	generated makefiles and support files
`install`	compiled binaries, libraries and installed headers
`log`	saved logs of previous development builds
`src`	package source codes
	place your stuff in here, do not modify the other folders

Navigating and compiling ROS workspace

finding a package	`$ ros2 pkg prefix --share [pkg]`
compile the whole workspace	`$ colcon build`
compile a particular package	`$ colcon build --packages-select [pkg]`
compile package + dependencies	`$ colcon build --packages-up-to [pkg]`
clean the workspace (extension)	`$ colcon clean workspace`
clean a particular package	`$ rm -rf build/[pkg] install/[pkg]`
show workspace config	`$ cat colcon_defaults.yaml`
show mixins (extension)	`$ colcon mixin list`
compile with mixins	`$ colcon build --mixin [profile]`
create a workspace (extension)	`$ colcon init`
source a workspace	`$ source [path]/setup.bash`

Would You Like to Know More? `https://colcon.readthedocs.io/`

ROS package structure

Some of the following items might be missing, depending on the package use case.

`package.xml`	manifest, dependencies and plugins
`CMakeLists.txt`	description of compilation procedure
`src/`	`C` and `C++` source codes
`include/`	`C` and `C++` headers
`scripts/`	Python and bash scripts
`config/`	yaml config files
`launch/`	ROS launch files

Would You Like to Know More?
`https://docs.ros.org/en/lyrical/Concepts/Advanced/About-Build-System.html`

ROS visualization tools

Rviz	3-D visualization of data and models
	`$ rviz2`
	`$ ros2 run rviz2 rviz2 -d ./rviz.rviz`

Rqt plot	simple and lightweight plotting `$ ros2 run rqt_plot rqt_plot`
Rqt bag	visualizing contents of a rosbag `$ ros2 run rqt_bag rqt_bag`
Plot juggler	complex and powerful plotting `$ ros2 run plotjuggler plotjuggler`
Rqt reconfigure	online parameter setting `$ ros2 run rqt_reconfigure rqt_reconfigure`
Rqt image view	camera images visualization `$ ros2 run rqt_image_view rqt_image_view`
Gazebo client	Gazebo GUI `$ gz sim`
rqt	Integrates most of the *rqt_* tools `$ rqt`

Would You Like to Know More? `https://docs.ros.org/en/lyrical/Tutorials/Beginner-CLI-Tools.html`

Useful UAV ROS topics and services

Following ROS services and topics allow for controlling the UAV from terminal. Each address contains a particular name of the UAV.

Informative topics (subscribe to know stuff)

state estimate (rviz-able)	`/uav1/estimation_manager/odom_main`
control reference (rviz-able)	`/uav1/control_manager/control_reference`
control reference (full-state)	`/uav1/control_manager/tracker_cmd`
control manager diagnostics	`/uav1/control_manager/diagnostics`

Control Services/Topics (call or publish to influence stuff)

The addresses for *reference*, *trajectory_reference* are the same for both the topic and the service.

position+heading goal	`/uav1/control_manager/reference`
takeoff	`/uav1/uav_manager/takeoff`
land	`/uav1/uav_manager/land`
land home	`/uav1/uav_manager/land_home`
hover	`/uav1/control_manager/hover`
switch controller	`/uav1/control_manager/switch_controller [Controller]`
switch tracker	`/uav1/control_manager/switch_tracker [Tracker]`
set tracker constraints	`/uav1/constraint_manager/set_constraints [Constraints]`
set SO(3) controller gains	`/uav1/gain_manager/set_gains [Gains]`
load trajectory	`/uav1/control_manager/trajectory_reference`
trajectory goto start	`/uav1/control_manager/goto_trajectory_start`
trajectory start tracking	`/uav1/control_manager/start_trajectory_tracking`

Would You Like to Know More? `https://ctu-mrs.github.io/docs/api/uav_ros_api`

SSH keys

- Generate your SSH key by: `$ ssh-keygen -t rsa -b 4096 -C "your_email@example.com"`.
- The keys are stored in `~/.ssh`.
- Show the content of the public key by: `$ cat ~/.ssh/id_rsa.pub` and copy it to Github or Gitlab.
- Copy your public key over ssh to another machine by: `$ ssh-copy-id user@machine`.
- Entries in the `~/.ssh/config` allow connecting to a machine via alias while using an ssh key:

```
host mrs
     hostname mrs.fel.cvut.cz
     user git
     identityfile ~/.ssh/id_rsa
```

Spawning a UAV in Gazebo simulator

We use a ROS node called `mrs_drone_spawner` to dynamically load a UAV into the Gazebo/ROS simulator. By default, it starts automatically with Gazebo using `$ ros2 launch mrs_uav_gazebo_simulator simulation.launch.py`.
Spawn a drone by calling the service `$ ros2 service call /mrs_drone_spawner/spawn mrs_msgs/srv/String`
If the service does not exist, start the spawner by `$ ros2 launch mrs_uav_gazebo_simulator mrs_drone_spawner.launch.py`
Various arguments can be used to influence the type of the drone, its sensors, its starting location and additional onboard hardware. Run the command `$ ros2 service call /mrs_drone_spawner/spawn mrs_msgs/srv/String "value: --x500 --help"` to see a list. Here are some notable examples:

use initial position from a CSV file (id, x, y, z, heading)	`--file [path_to_file]`
use initial position from an argument	`--pos [x y z heading]`
selecting UAV type (f450, f550, t650)	`--f450`, `--f550`, `--t650`
add down-facing rangefinder	`--enable-rangefinder`
add front-facing camera	`--enable-bluefox-camera`
add front-facing RealSense	`--enable-realsense-front`
add 2-D rangefinder	`--enable-rplidar`
add 3-D rangefinder	`--enable-velodyne`
add UV camera for UVDAR	`--enable-uv-camera`
add UV leds for UVDAR	`--enable-uv-leds`
set UV led frequencies (left)	`--uvled-fr-l [freq]`
add super long pendulum	`--enable-pendulum`
add ball holder	`--enable-ball-holder`

A typical simulation spawning looks like:
`$ ros2 service call /mrs_drone_spawner/spawn mrs_msgs/srv/String "value: 1 --f450 --enable-rangefinder"`

Zenoh on a remote machine

- Add your **local** machine hostname to the **remote** machine's hostname `/etc/hosts` and vice versa.
- Make sure the **robot's** `/etc/hosts` contains the '`127.0.1.1 <robot's hostname>`' entry.
- Make sure the machines can ping each other using their hostnames.
- Set matching `export ROS_DOMAIN_ID=<ID>` in the `~/.bashrc` on both machines (0 by default).
- Add `export RMW_IMPLEMENTATION=rmw_zenoh_cpp` to the `~/.bashrc` on both machines.
- Add `export ZENOH_CONNECT="tcp/<remote_host>:7447"` to the **local** machine's `~/.bashrc`.
- **Do NOT export** `ROS_DISCOVERY_SERVER` when using the Zenoh RMW.
- Run the Zenoh daemon `ros2 run rmw_zenoh_cpp rmw_zenohd` on both machines.

The math that everybody needs, but nobody remembers

2-D rotational matrix:

$$\mathbf{R}(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}$$

Degrees-to-radian conversion table with values of sin and cos:

deg	0	30	45	60	90	120	180
rad	0	0.523	0.785	1.047	1.57	2.09	3.14
sin	0.0	0.500	0.707	0.866	1.0	0.866	0.0
cos	1.0	0.866	0.707	0.500	0.0	-0.50	-1.0

Quaternions (unit quaternions)

"Complex" numbers with three imaginary parts: i, j, k and $\| \cdot \| = 1$.

By axis $[x, y, z]$ and angle ϕ	$q = \cos\frac{\phi}{2} + (xi + yj + zk)\sin\frac{\phi}{2}$
Component-wise	$q_w = \cos\frac{\phi}{2}$, $q_x = x\sin\frac{\phi}{2}$, $q_y = y\sin\frac{\phi}{2}$, $q_z = z\sin\frac{\phi}{2}$
Inverse quaternion	$q^{-1} = \cos\frac{-\phi}{2} + (xi + yj + zk)\sin\frac{-\phi}{2} = \frac{q_w - q_x i - q_y j - q_z k}{q_w^2 + q_x^2 + q_y^2 + q_z^2}$
Transforming the vector $[1, 2, 3]$	$u = 0 + 1i + 2j + 3k$, $v = quq^{-1}$

Converting various representations of rotation using `mrs_lib::AttitudeConverter`:

```
# every combination is possible
tf2::Quaternion                  tf2_quat       = AttitudeConverter(roll, pitch, yaw);
tf2::Matrix3x3                   tf2_matrix     = AttitudeConverter(tf2_quat);
geometry_msgs::msg::Quaternion   quaternion     = AttitudeConverter(tf2_matrix);
Eigen::Quaterniond               eig_quat       = AttitudeConverter(quaternion);
Eigen::AngleAxis<double>         eig_angle_axis = AttitudeConverter(eig_quat);
Eigen::Matrix3d                  eig_matrix     = AttitudeConverter(eig_angle_axis);
auto [roll2, pitch2, yaw2]                      = AttitudeConverter(eig_matrix);
std::tie(roll2, pitch2, yaw2)                   = AttitudeConverter(roll2, pitch2, yaw2);
double heading1                                 = AttitudeConverter(tf2_quat).getHeading();
```

Would You Like to Know More? `https://eater.net/quaternions`

Common ROS2 handlers in C++

node (standalone)	`auto node = rclcpp::Node::make_shared("node_name");`
component (nodelet)	`auto node = std::make_shared<rclcpp::Node>("node_name", options);`
subscriber	`auto sub = node->create_subscription<msg_class>("name", 1, callback);`
publisher	`auto pub = node->create_publisher<msg_class>("name", 1);`
service client	`auto client = node->create_client<srv_class>("name");`
service server	`auto server = node->create_service<srv_class>("name", callback);`
timer	`auto timer = node->create_wall_timer(std::chrono::duration<double>(1.0 / 30), callback);`

Would You Like to Know More? `https://docs.ros.org/en/lyrical/Tutorials.html`

Common ROS2 handlers in Python

node initialization	`rclpy.init(); node = rclpy.create_node('node_name')`
subscriber	`sub = node.create_subscription(MsgClass, 'topic_name', cb, 1)`
publisher	`pub = node.create_publisher(MsgClass, 'topic_name', 1)`
service client	`client = node.create_client(SrvClass, 'service_name')`
service server	`server = node.create_service(SrvClass, 'service_name', cb)`
timer	`timer = node.create_timer(1.0 / 30, cb)`

Would You Like to Know More? `https://docs.ros.org/en/lyrical/Tutorials.html`

Common Eigen operations in C++

Fixed matrix	`Matrix<double, 3, 3> A;`	element-wise product	`P.cwiseProduct(Q)`
Dynamic matrix	`MatrixXd A;`	Norm	`v.norm()`
Dynamic vector	`VectorXd v;`	Squared norm	`v.squaredNorm()`
Zero matrix	`MatrixXd::Zero(rows, cols)`	Dot product	`v.dot(u)`
Identity matrix	`MatrixXd::Identity(n, n)`	Cross product	`v.cross(v)`
Vector element	`v(n)`	Solve Ax=b	`x = A.qr().solve(b);`
Matrix element	`A(row, column)`	Eigen-decomposition	`EigenSolver<Matrix3d> eig(A);`
Matrix inversion	`A.inverse()`	Matrix transposition	`A.transpose()`
Matrix column	`A.col(n)`	`#include <Eigen/Dense>`	for everything
no. of rows and cols	`A.rows()`, `A.cols()`	`#include <Eigen/Geometry>`	for cross
Sub-matrix	`A.block(i, j, rows, cols)`	`#include <Eigen/QR>`	for QR decomposition

Would You Like to Know More? `https://eigen.tuxfamily.org/dox/AsciiQuickReference.txt`

GDB - GNU Debugger

If you're experiencing crashes of your `C/C++` ROS node/nodelet or if your program is not behaving as expected in general and you want to inspect it, you can reach for a debugger. A debugger (namely GDB in our case) enables you to inspect the state of the program after a crash or at any point during the program runtime and is a very powerful tool for rooting out bugs.

command	description	comment
`b filename.cpp:310`	breakpoint in *filename.cpp* at line 310	
`bt`	backtrace	
`f <num>`	change to frame `<num>`	`<num>` = the number from bt
`s`	step in function	
`n`	step to the next line	
`fin`	finish function	in case you accidentaly step into
`c`	continue	resume program until breakpoint or crash
`p <num>`	print variable	`<num>` = variable name
`wh`	open window with code (tui)	actually sets window height
`tui enable/disable`	open/close window with code (tui)	the official way of wh
`focus cmd/src`	changes focus in gdb tui	if you want to use arrows for cmd hist.
`up/down`	jumps in the frames ip/down	
`<enter>`	repeats the last command	
`u <num>`	continue until line `<num>`	`<num>` = the line number in the current file
`ref`	refresh the screen	in case of some visual problems
`run`	run the executable	
`thread apply all #`	apply command # to all threads	
the *.gdbinit* file	put pre-start settings in here	an example is in the file
–args exec arg1 ...	running executable with args	

Would You Like to Know More? `https://ctu-mrs.github.io/docs/software/cpp/cpp_debugging_with_gdb`

`mrs_lib::Transformer`, MRS `ros::tf2` wrapper

Although *ros::tf2* library provides options for transforming data between frames of reference, it is far from friendly-to-use. Therefore, we have built a wrapper that simplifies most of the tasks.

`include <mrs_lib/transformer.h>`	include this
`mrs_lib::Transformer transformer_;`	declare
`transformer_ = mrs_lib::Transformer(node_);`	initialize

- `mrs_lib::Transformer` is capable of inferring full name of a UAV: `gps_origin -> uav3/gps_origin` (after enabling by `transformer_.setDefaultPrefix(<uav_name>);`.
- `mrs_lib::Transformer` finds the latest available `<tf>` if there is none available for `<time>` (after enabling by `transformer_.retryLookupNewest(true)`
- `mrs_lib::Transformer` can transform from/to `latlon_origin`, our custom GPS frame with deg. of lat/lon

Getting transformation `<from>` frame `<to>` frame in particular `time`

```
if (auto ret = transformer_.getTransform(<from>, <to>, <time>) {
  mrs_lib::TransformStamped tf = ret.value();
}
```

Transforming `<what>` `<to>` frame using the transformation `<tf>`:

```
if (auto ret = transformer_.transform(<what>, <transformation>) {
  auto result = ret.value();
}
```

Transforming `<what>` `<to>` only once (finds the `<tf>` automatically):

```
if (auto ret = transformer_.transformSingle(<what>, <to>) {
  auto result = ret.value();
}
```

Would You Like to Know More? `https://ctu-mrs.github.io/mrs_lib`

https://github.com/ctu-mrs

https://mrs.fel.cvut.cz

this cheat sheet PDF

`https://github.com/ctu-mrs/mrs_cheatsheet`

vim

[operator][count][motion]

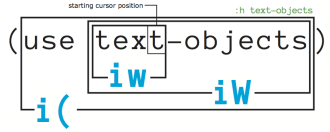
:h operator — :h navigation

d delete/cut
y yank/copy
c change
gU make uppercase · ~ swap case · < shift left · = indent

Any motion can follow an operator. Marks and searches count as motions, too! `d/foo` will delete from the cursor to the next instance of "foo". `y3fi` will yank from the cursor to the 3rd "i" on the line after it. Counts can also come before operators: `5dd` will delete five lines.

w	word
W	WORD
s	sentence
[,]	[ ] block
(,)	( ) block
< , >	< > block
t	XML/HTML tag
{ , }	{ } block
" , '	quoted string

:h text-objects



beginning of line **0** · first non-blank character **^** · previous WORD **B** · previous word **b** · previous character **h**

gg first line · **^b** up 1 page · **^u** up ½ page · **k** up 1 line

:h up-down-motions

	ts	sw	sts	et
use spaces only	*n*	*n*	*n*	on
use tabs only	*n*	*n*	0	off

tabstop	ts	Columns per tabstop
shiftwidth	sw	Columns per `<<`
softtabstop	sts	Spaces per tab
expandtab	et	`<Tab>` inserts spaces

Set *n* to desired tab width (default 8)

MIXING TABS AND SPACES IS RIGHT OUT. (that means don't do it.)

:retab Replace all tabs with spaces according to current tabstop setting

fileformat	ff	Try changing this if your line-endings are messed up
list		Display whitespace visibly according to listchars

:h left-right-motions

next character **l** · end of word **e** · beginning of next word **w** · end of WORD **E** · beginning of next WORD **W** · end of line **$**

j down 1 line · **^d** down ½ page · **^f** down 1 page · **G** last line

:help

:h *cmd*	Normal mode *cmd* help
:h i_*cmd*	Insert mode *cmd* help
:h v_*cmd*	Visual mode *cmd* help
:h c_*cmd*	Command-line editing *cmd* help
:h :*cmd*	Command-line *cmd* help
:h 'option'	*Option* help
:helpgrep	Search through all help docs!

:h tags-and-searches

^]	Jump to tag under cursor, including [tags] in help files
^t	Jump back up the tag-list
g^]	Jump to tag if it's the only match; else list matching tags

:h keycodes

<CR>	^m	\r	Enter
<Tab>	^i	\t	Tab
<C-*n*>	^n		Ctrl-*n*
<M-*n*>			Alt-*n*
<Esc>	^[		Escape
<BS>	^h	\b	Backspace
<Del>			Delete

7 words :h word-motions

http://www.vimcheatsheet.com

1 WORD

SEARCHING

:h pattern-searches

Prev	Next	Forward	Backward	Matches
N	n	*/foo*	*?foo*	*foo*
		*	#	word under cursor
;	,	t*x*	T*x*	upto *x*
		f*x*	F*x*	find *x*

:h mark-motions

m*m*	set mark *m* (a-z) in file	**m*M***	set mark *M* (A-Z) across files
'[	jump to first char of just-changed text		
'*m*	jump to first char of line containing *m*	**`*m***	jump to exact character of *m*
''	jump back to last jump		

p	paste after cursor	**P**	paste before cursor	**^[**	return to Normal mode
u	undo	**^r**	redo	**.**	repeat
gf	find file under cursor in path and jump to it (file-searching)	**dd**	delete current line	**yy**	yank current line
x	delete character after cursor	**%**	jump to matching paren	**r**	replace char under cursor
***n*G**	jump to line *n*	**^o**	jump back (jump-motions)	**^i**	jump forward (jump-motions)
zz	center screen on cursor	**zt**	align top of screen with cursor	**zb**	align bottom of screen with cursor
==	auto-indent current line	**<<**	shift current line left by shiftwidth	**>>**	shift current line right by shiftwidth

Pass a directory to the `:edit` command to open a directory explorer. Instructions for usage are at the top of the screen.

Using `^[` to return to Normal mode lets you keep your fingers on the home row. It's even easier if you map Caps Lock to Control!

ENTERING INSERT MODE

:h insert.txt

beginning of line **I** · before cursor **i** · after cursor **a** · end of line **A**
previous line **O** · next line **o** · substitute character **s** · substitute line **S** · line from cursor **C**

COOL INSERT MODE STUFF

^w	delete word before cursor	**^u**	delete line before cursor
^r*r*	insert the contents of register *r*	**^r=**	use the expression register (try `^r=5+10`)
^t	increase line indent by shiftwidth	**^d**	decrease line indent by shiftwidth
^x^l	line completion	**^n**	find next completion suggestion according to complete

ENTERING VISUAL (SELECT) MODE

:h cmdline.txt

v The most basic type. Use visual mode to select characters within a line.

V Useful for moving chunks of a program around the file. Use Visual Line mode to select one or more lines.

^v Great for working with tables made of text, or anything that happens to be conveniently aligned. Visual Block mode can be used to select boxes across lines.

switch cursor to start/end **o** :h v_o · re-select previous area **gv** :h gv · prepend to each Visual block line **I** :h v_b_I · jump to start of prior area **'<** :h '<

COMMAND-LINE MODE ONLY

edit using Normal mode **^f** cmdwin · insert word under cursor **^r^w** cmdline-editing · completion suggestions **^d** cmdline-completion

Put `cnoremap %% <C-R>=expand('%:h').'/'<CR>` in your `vimrc` so you can type `%%` in Command-line mode to refer to the directory of the current file, regardless of `pwd`.

ZZ	Write current file, if modified, and quit	
ZQ	Quit without checking for changes (like `:q!`)	
:write	Write current file	
:wq	Write current file and quit	

Use `:scriptnames` to list all files sourced during initialization.

:syntax	Enable and configure syntax highlighting Use `:sy sync fromstart` to redraw broken highlights
:make	Run a compiler and enter quickfix mode (:h quickfix)
:!	Execute external shell command
!	Filter motion with shell command

Use `:earlier` and `:later` to quickly jump backward and forward in a file's history.

:read	Read external program output into current file

Supply `%` as a range to the `:substitute` command to run it on every line in the file.

`:%s/Scribbl/Design/`	"Scribbled" -> "Designed"

Specify the "g" flag to apply the substitution to *every* match on a line.

`:s/[dla]//g`	"badly" -> "by" (:h s_flags, :h /[])

Vim supports many regular expression features.

`:s/..k/ax/`	"Mook" -> "Max" (:h usr_27, :h /.)

Use `\_.` instead of `.` if you want to search across multiple lines.

`:%s/heat\_.*Bungle/anto/`	"Cheatsheet\nBungler" -> "Cantor" (:h /_.)

Special escapes can be used to change the case of substitutions.

`:s_\(f..\)_\U\1\E_`	"foobar" -> "FOObar" (:h sub-replace-special)

Use `:global` to perform a command on matching lines.

`:g/foobar/delete`	Delete all lines containing "foobar"

If your pattern contains slashes, just use a different character as your delimiter.

`:s_Data/Lore_Brent Spiner_`

Use `\=` to evaluate expressions with replacement groups.

`:s_\d_\=submatch(0) + 1_g`	"10 25" -> "21 36" (:h sub-replace-\=)

:options

:set *opt*?	View current value of *opt*
:set no*opt*	Turn off flag *opt*
:set *opt*	Turn on flag *opt*
:set *opt*=*val*	Overwrite value of *opt*
:set *opt*+=*val*	Append to value of *opt*
:echo &*opt*	Access *opt* as a variable

hidden	hid	Lets you switch buffers without saving
laststatus	ls	Show status line never (0), always (2) or with 2+ windows (1)
hlsearch	hls	Highlight search matches. Also see 'highlight'
number	nu	Show line numbers
showcmd	sc	Show commands as you type them
ruler	ru	Show line and column number of the cursor
backspace	bs	Set to '2' to make backspace work like sane editors
wrap		Control line wrapping
background	bg	Set to 'dark' if you have a dark color scheme

Use `a` instead of `i` when beginning text-object motions to include delimiters or surrounding whitespace. For example, `di(` will change "(foo)" into "()", but `da(` will delete the parentheses as well.

:buffers

:ls	List all open files
:b *path*	Jump to unique file matching *path*. Use **<Tab>** to scroll through available completions!
:bn	Jump to file *n*, number from first column of :ls
:bnext	Jump to next file
:bprev	Jump to previous file
:bdelete	Remove file from the buffer list
:edit	Open a file for editing
:enew	Open a blank new file for editing

:windows

:split	Split current window horizontally
:vsplit	Split current window vertically
^w hjkl	Move cursor to window left, below, above or to the right of the current window
^w HJKL	Move current window to left, bottom, top, or right of screen
^w r	Rotate windows clockwise
^w +-<>	Increase/decrease current window height/width
^w T	Move current window to a new tab
:only	Close all windows except current window
:bufdo	Execute a command in each open file

REGISTERS are CLIPBOARDS

All commands that delete, copy, or paste text use registers. To change which register is used by a command, type the register before the command. The default register is called "the unnamed register", and it is invoked with a pair of double-quotes (`""`). Typing `dd` or `yy` is the same as typing `""dd` or `""yy`. Think of the first `"` as a short way of saying "register", so `""` is pronounced "register "", and `"a` "register a".

:h registers

:registers	View all current registers
:echo @*r*	Access register *r* as a variable
"/	Last search pattern register — Contains the last pattern you searched for
"_	The black hole register — Use this to delete without clobbering any register (`"_dd`)
"0	Last yank register — Contains the last text you yanked
"1	Last big delete register — Contains the last line(s) you deleted
"2-"9	Big delete register stack — Every time `"1` is written to, its content is pushed to `"2`, then `"2` to `"3`, and so on
"-	Small delete register — Contains the last text you deleted within a single line
"+	System clipboard — If the OS integration gods smile upon you, this register reads and writes to your system clipboard.
"a-"z	Named registers — 26 registers for you to play with
"A-"Z	Append registers — Using upper-case to refer to a register will append to it rather than overwrite it
q*r*	Record — Record into register *r*. Stop recording by hitting `q` again
@*r*	Playback — Execute the contents of register *r*
@@	Repeat last playback — Repeat the last `@r`, this is particularly useful with a count

Use `:map` to view all current custom key mappings. Read `:h map-which-keys` for a guide on which keys are best for your own custom mappings. Get used to Vim's help system - it's a fantastic resource!

vim one-liner used to sort the list of names by length:
`:exe 'g/^/let @x = len(getline("."))' | normal "xPa ' ' | sort n | :g//normal dw`

1st Edition
funded by...
MML
Adam.
jdiemz
Lin Qiu
@webjoe
grantok
storrgie
David O.
Digininja
Dave Remy
Ralph Bean
Arve Løken
Rajat Dhal
Paul-Kenji
Mark Dixon
Neil Munro
Kyle Verma
Guy Bensky
Ross Timson
Abdul Qabiz
Daniel Seck
Hazel Smith
Kelly Rails
Charles Chan
Pivotal Labs
Eric Sporkin
M. Adam Price
Daniel Kehler
Rauli Silvola
Lawrence Kemp
Bill LaPierre
Brandon Tyree
Laurence Emms
Infonautic.de
@anderssothus
Damon Jablons
Travis McHenry
Ganesh Sugunan
Rouven Hornler
Digital Gnomes
San Napolitano
Jens O. Meiert
Thomas Nitsche
Philip Ratzesch
Hurricane Labs
Michael Schmidt
Matthew Datcher
Arya Reais-Parsi
David Rochefarine
Christer Edvartsen
St. Louis Vim Geeks
Haraldur Tristan Gunnarsson
and others!