Maven cheat sheet

Getting started with Maven

Create Java project

```
mvn archetype:generate
-DgroupId=org.yourcompany.project
-DartifactId=application
```

Create web project

```
mvn archetype:generate
-DgroupId=org.yourcompany.project
-DartifactId=application
-DarchetypeArtifactId=maven-archetype-webapp
```

Create archetype from existing project

```
mvn archetype:create-from-project
```

Main phases

clean — delete target directory
validate — validate, if the project is correct
compile — compile source code, classes stored in target/classes
test — run tests
package — take the compiled code and package it in its distributable format, e.g. JAR, WAR
verify — run any checks to verify the package is valid and meets quality criteria
install — install the package into the local repository
deploy — copies the final package to the remote repository

Useful command line options

-DskipTests=true compiles the tests, but skips running them

-Dmaven.test.skip=true skips compiling the tests and does not run them

-T - number of threads:
-T 4 is a decent default
-T 2C - 2 threads per CPU

-rf, --resume-from resume build from the specified project

-pl, --projects makes Maven build only specified modules and not the whole project

-am, --also-make makes Maven figure out what modules out target depends on and build them too

-o, --offline work offline

-X, --debug enable debug output

-P, --activate-profiles comma-delimited list of profiles to activate

-U, --update-snapshots forces a check for updated dependencies on remote repositories

-ff, --fail-fast stop at first failure

Essential plugins

Help plugin — used to get relative information about a project or the system.
mvn help:describe describes the attributes of a plugin
mvn help:effective-pom displays the effective POM as an XML for the current build, with the active profiles factored in.

Dependency plugin — provides the capability to manipulate artifacts.
mvn dependency:analyze analyzes the dependencies of this project
mvn dependency:tree prints a tree of dependencies

Compiler plugin — compiles your java code.
Set language level with the following configuration:

```
<plugin>
 <groupId>org.apache.maven.plugins</groupId>
 <artifactId>maven-compiler-plugin</
  artifactId>
 <version>3.6.1</version>
 <configuration>
   <source>1.8</source>
   <target>1.8</target>
 </configuration>
</plugin>
```

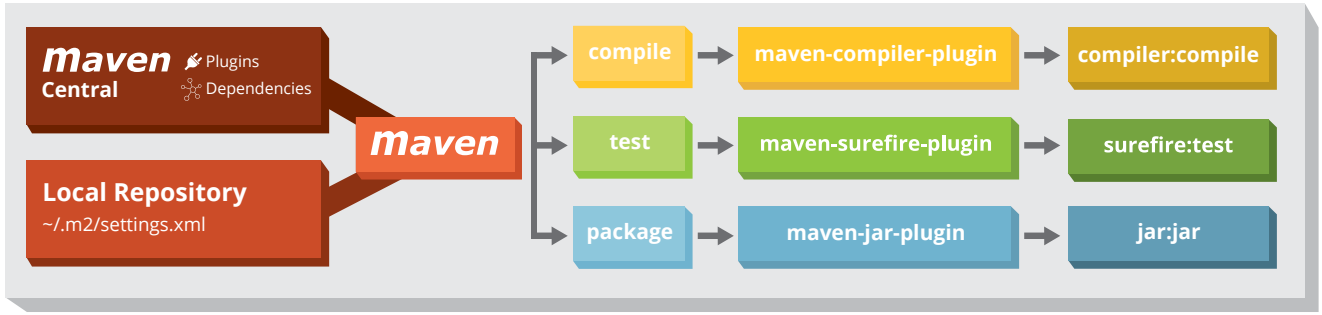
Version plugin — used when you want to manage the versions of artifacts in a project's POM.

Wrapper plugin — an easy way to ensure a user of your Maven build has everything that is necessary.

Spring Boot plugin — compiles your Spring Boot app, build an executable fat jar.

Exec — amazing general purpose plugin, can run arbitrary commands :)

The big picture



Git Cheat Sheet

For more awesome cheat sheets visit rebellabs.org!

REBELLABS by ZEROTURNAROUND

Create a Repository

From scratch -- Create a new local repository

```
$ git init [project name]
```

Download from an existing repository

```
$ git clone my_url
```

Observe your Repository

List new or modified files not yet committed

```
$ git status
```

Show the changes to files not yet staged

```
$ git diff
```

Show the changes to staged files

```
$ git diff --cached
```

Show all staged and unstaged file changes

```
$ git diff HEAD
```

Show the changes between two commit ids

```
$ git diff commit1 commit2
```

List the change dates and authors for a file

```
$ git blame [file]
```

Show the file changes for a commit id and/or file

```
$ git show [commit]:[file]
```

Show full change history

```
$ git log
```

Show change history for file/directory including diffs

```
$ git log -p [file/directory]
```

Working with Branches

List all local branches

```
$ git branch
```

List all branches, local and remote

```
$ git branch -av
```

Switch to a branch, my_branch, and update working directory

```
$ git checkout my_branch
```

Create a new branch called new_branch

```
$ git branch new_branch
```

Delete the branch called my_branch

```
$ git branch -d my_branch
```

Merge branch_a into branch_b

```
$ git checkout branch_b
$ git merge branch_a
```

Tag the current commit

```
$ git tag my_tag
```

Make a change

Stages the file, ready for commit

```
$ git add [file]
```

Stage all changed files, ready for commit

```
$ git add .
```

Commit all staged files to versioned history

```
$ git commit -m "commit message"
```

Commit all your tracked files to versioned history

```
$ git commit -am "commit message"
```

Unstages file, keeping the file changes

```
$ git reset [file]
```

Revert everything to the last commit

```
$ git reset --hard
```

Synchronize

Get the latest changes from origin (no merge)

```
$ git fetch
```

Fetch the latest changes from origin and merge

```
$ git pull
```

Fetch the latest changes from origin and rebase

```
$ git pull --rebase
```

Push local changes to the origin

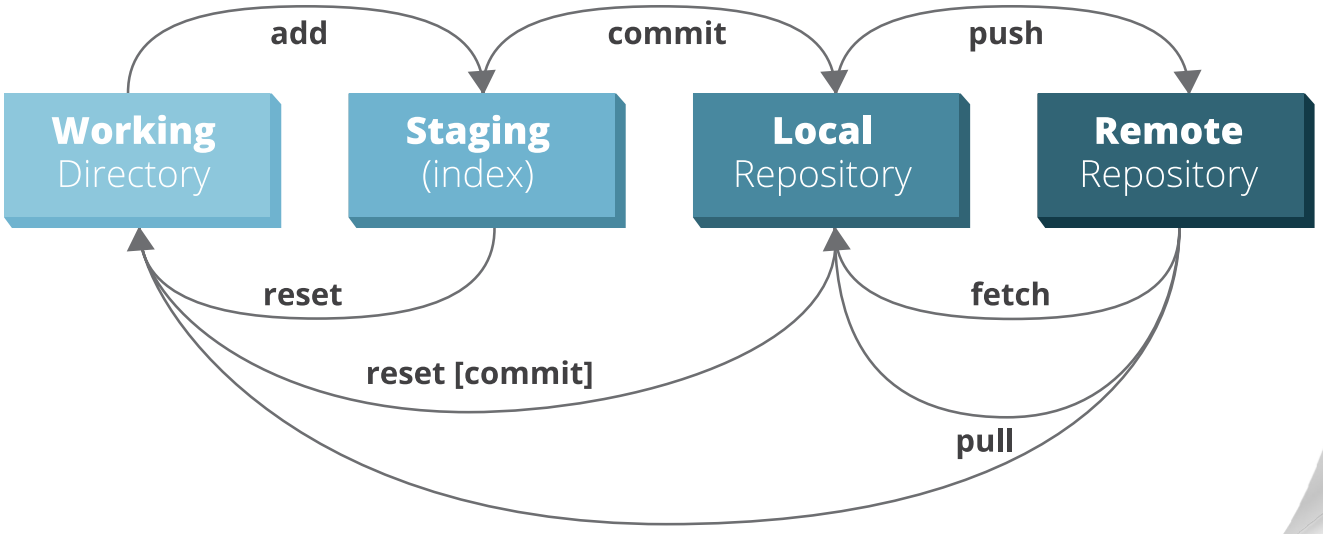
```
$ git push
```

Finally!

When in doubt, use git help

```
$ git command --help
```

Or visit https://training.github.com/ for official GitHub training.



Java Collections Cheat Sheet

For more awesome cheat sheets visit rebellabs.org!

REBELLABS by ZEROTURNAROUND

Notable Java collections libraries

Fastutil

http://fastutil.di.unimi.it/

Fast & compact type-specific collections for Java

Great default choice for collections of primitive types, like int or long. Also handles big collections with more than 2^{31} elements well.

Guava

https://github.com/google/guava

Google Core Libraries for Java 6+

Perhaps the default collection library for Java projects. Contains a magnitude of convenient methods for creating collection, like fluent builders, as well as advanced collection types.

Eclipse Collections

https://www.eclipse.org/collections/

Features you want with the collections you need

Previously known as gs-collections, this library includes almost any collection you might need: primitive type collections, multimaps, bidirectional maps and so on.

JCTools

https://github.com/JCTools/JCTools

Java Concurrency Tools for the JVM.

If you work on high throughput concurrent applications and need a way to increase your performance, check out JCTools.

What can your collection do for you?

Collection class	Thread-safe alternative	Your data				Operations on your collections						
						Order of iteration				Random access		
		Individual elements	Key-value pairs	Duplicate element support	Primitive support	FIFO	Sorted	LIFO	Performant 'contains' check	By key	By value	By index
HashMap	ConcurrentHashMap	✗	✓	✗	✗	✗	✗	✗	✓	✓	✗	✗
HashBiMap (Guava)	Maps.synchronizedBiMap (new HashBiMap())	✗	✓	✗	✗	✗	✗	✗	✓	✓	✓	✗
ArrayListMultimap (Guava)	Maps.synchronizedMultiMap (new ArrayListMultimap())	✗	✓	✓	✗	✗	✗	✗	✓	✓	✗	✗
LinkedHashMap	Collections.synchronizedMap (new LinkedHashMap())	✗	✓	✗	✗	✓	✗	✗	✓	✓	✗	✗
TreeMap	ConcurrentSkipListMap	✗	✓	✗	✗	✗	✓	✗	✓*	✓*	✗	✗
Int2IntMap (Fastutil)		✗	✓	✗	✓	✗	✗	✗	✓	✓	✗	✓
ArrayList	CopyOnWriteArrayList	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗	✓
HashSet	Collections.newSetFromMap (new ConcurrentHashMap<>())	✓	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗
IntArrayList (Fastutil)		✓	✗	✓	✓	✓	✗	✓	✗	✗	✗	✓
PriorityQueue	PriorityBlockingQueue	✓	✗	✓	✗	✗	✓**	✗	✗	✗	✗	✗
ArrayDeque	ArrayBlockingQueue	✓	✗	✓	✗	✓**	✗	✓**	✗	✗	✗	✗

** O(log(n)) complexity, while all others are O(1) - constant time*

*** when using Queue interface methods: offer() / poll()*

How fast are your collections?

Collection class	Random access by index / key	Search / Contains	Insert
ArrayList	O(1)	O(n)	O(n)
HashSet	O(1)	O(1)	O(1)
HashMap	O(1)	O(1)	O(1)
TreeMap	O(log(n))	O(log(n))	O(log(n))

Remember, not all operations are equally fast. Here's a reminder of how to treat the Big-O complexity notation:

O(1) - constant time, really fast, doesn't depend on the size of your collection

O(log(n)) - pretty fast, your collection size has to be extreme to notice a performance impact

O(n) - linear to your collection size: the larger your collection is, the slower your operations will be

Java Generics cheat sheet

Basics

Generics don't exist at runtime!

```
class Pair<T1, T2> { /* ... */ }
```
-- the type parameter section, in angle brackets, specifies type variables.

Type parameters are substituted when objects are instantiated.

```
Pair<String, Long> p1 = new
  Pair<String, Long> ("RL", 43L);
```

Avoid verbosity with the diamond operator:

```
Pair<String, Long> p1 =
  new Pair<>("RL", 43L);
```

Wildcards

`Collection<Object>` - heterogenous, any object goes in.
`Collection<?>` - homogenous collection of arbitrary type.

Avoid using wildcards in return types!

Intersection types

```
<T extends Object &
  Comparable<? super T>> T
max(Collection<? extends T> coll)
```

The return type here is ***Object***!

Compiler generates the bytecode for the most general method only.

Producer Extends Consumer Super (PECS)

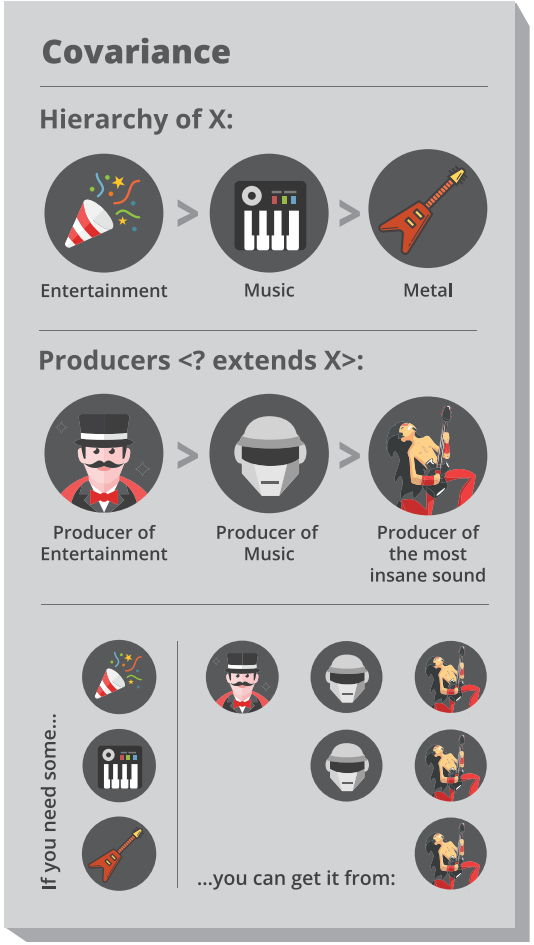
```
Collections.copy(List<? super T> dest, List<? extends T> src)
```

src -- contains elements of type T or its subtypes.

dest -- accepts elements, so defined to use T or its supertypes.

*Consumers are **contravariant** (use super).*

*Producers are **covariant** (use extends).*



Method Overloading

```
String f(Object s) {
  return "object";
}
String f(String s) {
  return "string";
}
<T> String generic(T t) {
  return f(t);
}
```

If called **generic("string")** returns "object".

Recursive generics

Recursive generics add constraints to your type variables. This helps the compiler to better understand your types and API.

```
interface Cloneable<T extends
Cloneable<T>> {
  T clone();
}
```

Now `cloneable.clone().clone()` will compile.

Covariance

```
List<Number> ⊁ ArrayList<Integer>
```

Collections are not covariant!

Java 8 Best Practices Cheat Sheet

For more awesome cheat sheets visit rebellabs.org!

REBELLABS by ZEROTURNAROUND

BROUGHT TO YOU BY JRebel

Default methods

Evolve interfaces & create traits

```
//Default methods in interfaces
@FunctionalInterface
interface Utilities {
  default Consumer<Runnable> m() {
    return (r) -> r.run();
  }
  // default methods, still functional
  Object function(Object o);
}
class A implements Utilities { // implement
  public Object function(Object o) {
    return new Object();
  }
  {
    // call a default method
    Consumer<Runnable> n = new A().m();
  }
}
```

Lambdas

Syntax:
(parameters) -> expression
(parameters) -> { statements; }

```
// takes a Long, returns a String
Function<Long, String> f = (l) -> l.toString();
// takes nothing gives you Threads
Supplier<Thread> s = Thread::currentThread;
// takes a string as the parameter
Consumer<String> c = System.out::println;

// use them with streams
new ArrayList<String>().stream().
// peek: debug streams without changes
peek(e -> System.out.println(e)).
// map: convert every element into something
map(e -> e.hashCode()).
// filter: pass some elements through
filter (hc -> (hc % 2) == 0).
// collect all values from the stream
collect(Collectors.toCollection(TreeSet::new))
```

java.util. Optional

A container for possible null values

```
// Create an optional
Optional<String> optional =
Optional.ofNullable(a);
// process the optional
optional.map(s -> "RebelLabs:" + s);
// map a function that returns Optional
optional.flatMap(s -> Optional.ofNullable(s));

// run if the value is there
optional.ifPresent(System.out::println);
// get the value or throw an exception
optional.get();

// return the value or the given value
optional.orElse("Hello world!");
// return empty Optional if not satisfied
optional.filter(s -> s.startsWith("RebelLabs"));
```

Rules of Thumb

Traits: 1 default method per interface
Don't enhance functional interfaces
Only conservative implementations

Expressions over statements
Refactor to use method references
Chain lambdas rather than growing them

Fields - use plain objects
Method parameters, use plain objects
Return values - use Optional
Use *orElse()* instead of *get()*

Java 8 Streams Cheat Sheet

For more awesome cheat sheets visit rebellabs.org!

REBELLABS by ZEROTURNAROUND

Definitions

- ☑ A stream **is** a pipeline of functions that can be evaluated.
- ☑ Streams **can** transform data.
- ☒ A stream **is not** a data structure.
- ☒ Streams **cannot** mutate data.

Intermediate operations

- Always return streams.
- Lazily executed.

Common examples include:

Function	Preserves count	Preserves type	Preserves order
map	✓	✗	✓
filter	✗	✓	✓
distinct	✗	✓	✓
sorted	✓	✓	✗
peek	✓	✓	✓

Stream examples

Get the unique surnames in uppercase of the first 15 book authors that are 50 years old or over.

```
library.stream()
     .map(book -> book.getAuthor())
     .filter(author -> author.getAge() >= 50)
     .distinct()
     .limit(15)
     .map(Author::getSurname)
     .map(String::toUpperCase)
     .collect(toList());
```

Compute the sum of ages of all female authors younger than 25.

```
library.stream()
     .map(Book::getAuthor)
     .filter(a -> a.getGender() == Gender.FEMALE)
     .map(Author::getAge)
     .filter(age -> age < 25)
     .reduce(0, Integer::sum):
```

Terminal operations

- Return concrete types or produce a side effect.
- Eagerly executed.

Common examples include:

Function	Output	When to use
reduce	concrete type	to cumulate elements
collect	list, map or set	to group elements
forEach	side effect	to perform a side effect on elements

Parallel streams

Parallel streams use the common ForkJoinPool for threading.

```
library.parallelStream()...
```

or intermediate operation:

```
IntStream.range(1, 10).parallel()...
```

Useful operations

Grouping:

```
library.stream().collect(
     groupingBy(Book::getGenre));
```

Stream ranges:

```
IntStream.range(0, 20)...
```

Infinite streams:

```
IntStream.iterate(0, e -> e + 1)...
```

Max/Min:

```
IntStream.range(1, 10).max();
```

FlatMap:

```
twitterList.stream()
     .map(member -> member.getFollowers())
     .flatMap(followers -> followers.stream())
     .collect(toList());
```

Pitfalls

☒ Don't update shared mutable variables i.e.

```
List<Book> myList =
new ArrayList<>();
library.stream().forEach
(e -> myList.add(e));
```

☒ Avoid blocking operations when using parallel streams.

JVM Options cheat sheet

For more awesome cheat sheets visit rebellabs.org!

REBELLABS by ZEROTURNAROUND

Standard Options

`$ java`
List all standard options.

`-Dblog=RebelLabs`
Sets a 'blog' system property to 'RebelLabs'. Retrieve/set it during runtime like this:

```
System.getProperty("blog");
//RebelLabs

System.setProperty("blog", "RL");
```

`-javaagent:/path/to/agent.jar`
Loads the java agent in agent.jar.

`-agentpath:pathname`
Loads the native agent library specified by the absolute path name.

`-verbose:[class/gc/jni]`
Displays information about each loaded class/gc event/JNI activity.

Non-Standard Options

`$ java -X`
List all non-standard options.

`-Xint`
Runs the application in interpreted-only mode.

`-Xbootclasspath:path`
Path and archive list of boot class files.

`-Xloggc:filename`
Log verbose GC events to filename.

`-Xms1g`
Set the initial size (in bytes) of the heap.

`-Xmx8g`
Specifies the max size (in bytes) of the heap.

`-Xnoclassgc`
Disables class garbage collection.

`-Xprof`
Profiles the running program.

Advanced Options

BEHAVIOR

`-XX:+UseConcMarkSweepGC`
Enables CMS garbage collection.

`-XX:+UseParallelGC`
Enables parallel garbage collection.

`-XX:+UseSerialGC`
Enables serial garbage collection.

`-XX:+UseG1GC`
Enables G1GC garbage collection.

`-XX:+FlightRecorder` (requires **`-XX:+UnlockCommercialFeatures`**)
Enables the use of the Java Flight Recorder.

DEBUGGING

`-XX:ErrorFile=file.log`
Save the error data to file.log.

`-XX:+HeapDumpOnOutOfMemory`
Enables heap dump when **`OutOfMemoryError`** is thrown.

`-XX:+PrintGC`
Enables printing messages during garbage collection.

`-XX:+TraceClassLoading`
Enables Trace loading of classes.

`-XX:+PrintClassHistogram`
Enables printing of a class instance histogram after a **`Control+C`** event (**`SIGTERM`**).

PERFORMANCE

`-XX:MaxPermSize=128m` (Java 7 or earlier)
Sets the max perm space size (in bytes).

`-XX:ThreadStackSize=256k`
Sets Thread Stack Size (in bytes).
(Same as -Xss256k)

`-XX:+UseStringCache`
Enables caching of commonly allocated strings.

`-XX:G1HeapRegionSize=4m`
Sets the sub-division size of G1 heap (in bytes).

`-XX:MaxGCPauseMillis=n`
Sets a target for the maximum GC pause time.

`-XX:MaxNewSize=256m`
Max size of new generation (in bytes).

`XX:+AggressiveOpts`
Enables the use of aggressive performance optimization features.

`-XX:OnError="<cmd args>"`
Run user-defined commands on fatal error.

Regex cheat sheet

Character classes

`[abc]`	matches **a** or **b**, or **c**.
`[^abc]`	negation, matches everything except **a**, **b**, or **c**.
`[a-c]`	range, matches **a** or **b**, or **c**.
`[a-c[f-h]]`	union, matches **a**, **b**, **c**, **f**, **g**, **h**.
`[a-c&&[b-c]]`	intersection, matches **b** or **c**.
`[a-c&&[^b-c]]`	subtraction, matches **a**.

Predefined character classes

`.`	Any character.
`\d`	A digit: `[0-9]`
`\D`	A non-digit: `[^0-9]`
`\s`	A whitespace character: `[ \t\n\x0B\f\r]`
`\S`	A non-whitespace character: `[^\s]`
`\w`	A word character: `[a-zA-Z_0-9]`
`\W`	A non-word character: `[^\w]`

Boundary matches

`^`	The beginning of a line.
`$`	The end of a line.
`\b`	A word boundary.
`\B`	A non-word boundary.
`\A`	The beginning of the input.
`\G`	The end of the previous match.
`\Z`	The end of the input but for the final terminator, if any.
`\z`	The end of the input.

Pattern flags

`Pattern.CASE_INSENSITIVE` - enables case-insensitive matching.

`Pattern.COMMENTS` - whitespace and comments starting with **#** are ignored until the end of a line.

`Pattern.MULTILINE` - one expression can match multiple lines.

`Pattern.UNIX_LINES` - only the '**\n**' line terminator is recognized in the behavior of **.**, **^**, and **$**.

Useful Java classes & methods

PATTERN

A pattern is a compiler representation of a regular expression.

`Pattern compile(String regex)`
Compiles the given regular expression into a pattern.

`Pattern compile(String regex, int flags)`
Compiles the given regular expression into a pattern with the given flags.

`boolean matches(String regex)`
Tells whether or not this string matches the given regular expression.

`String[] split(CharSequence input)`
Splits the given input sequence around matches of this pattern.

`String quote(String s)`
Returns a literal pattern String for the specified String.

`Predicate<String> asPredicate()`
Creates a predicate which can be used to match a string.

MATCHER

An engine that performs match operations on a character sequence by interpreting a Pattern.

`boolean matches()`
Attempts to match the entire region against the pattern.

`boolean find()`
Attempts to find the next subsequence of the input sequence that matches the pattern.

`int start()`
Returns the start index of the previous match.

`int end()`
Returns the offset after the last character matched.

Quantifiers

Greedy	Reluctant	Possessive	Description
X?	X??	X?+	*X, once or not at all.*
X*	X*?	X*+	*X, zero or more times.*
X+	X+?	X++	*X, one or more times.*
X{n}	X{n}?	X{n}+	*X, exactly n times.*
X{n,}	X{n,}?	X{n,}+	*X, at least n times.*
X{n,m}	X{n,m}?	X{n,m}+	*X, at least n but not more than m times.*

Greedy - matches the longest matching group.
Reluctant - matches the shortest group.
Possessive - longest match or bust (no backoff).

Groups & backreferences

A group is a captured subsequence of characters which may be used later in the expression with a backreference.

`(...)` - defines a group.
`\N` - refers to a matched group.

`(\d\d)` - a group of two digits.
`(\d\d)/\1` - two digits repeated twice.
`\1` - refers to the matched group.

Logical operations

XY	**X** then **Y**.
X|Y	**X** or **Y**.

Markdown Cheat Sheet

Ralph Mason, Author

Headings (Core Markdown)

```
# This is an H1
## This is an H2
### This is an H3
###### This is an H6
```

or...

```
This is an H1
========
This is an H2
--------
```

Ordered Lists (Core Markdown)

```
1. Top-level item
2. Top-level item
3. Top-level item
  1. Nested item
  2. Nested item
```

• You must include a **.** after each number
• Markdown will output items in order (1, 2, 3...) even if you write unorderly (1, 3, 2...)

Text Formatting (Core Markdown)

```
*This is em text* and _so is this_
**This is strong text** and __so is this__
*This is **strong text** inside em text*
**This is _em text_ inside strong text**
***This is strong and em combined***
```

Paragraphs (Core Markdown)

Paragraphs don't require markup, but make sure:

• There's a line break before and after it
• There are no spaces or tabs at the start of it

Unordered Lists (Core Markdown)

```
* Top-level item
* Top-level item
  * Nested item
```

• Optionally use **+** or **-** instead of ***
• Indent nested items by 2 spaces
• Lists must have a line break before and after
• List *items* can be indented by max. three spaces or one tab
• Hanging indents for wrapped lines are supported:

```
    - This is nicer to read
      in plain text
```

Paragraphs in List Items (Core Markdown)

Adding line breaks between list items will wrap the list item content in **<p>** tags.

```
- item 1

- item 2
```

For *multiple paragraphs* in a list item, indent each one by four spaces or a tab (first item optional).

```
-   Paragraph 1

    Paragraph 2
```

Markdown Cheat Sheet

Ralph Mason, Author

Code Blocks (Core Markdown)

Place three backticks above and below the block

````
```
.selector {
  color: red;
}
```
````

Or, indent every line by at least 4 spaces/one tab

```
    .selector {
      color: red;
    }
```

- Markdown renders code inside **<pre><code>** tags
- Place one line break above/below the code block

Code Block in a List Item (Core Markdown)

Indent the code block by 8 spaces or 2 tabs

```
* list item:

        console.log("A code block")
```

Inline Code (Core Markdown)

Wrap inline code with single backticks

```
Run `console.log("Hello world");` in the console.
```

Blockquotes (Core Markdown)

```
> One paragraph in
  a blockquote

> Another paragraph in
  the same blockquote.
```

Alternatively, use **>** before every line.

```
> One paragraph in
> a blockquote
>
> Another paragraph in
> the same blockquote.
```

Nested Blockquotes (Core Markdown)

```
> A top-level blockquote
>
>> A nested blockquote
>
> Top-level blockquote continued.
```

Elements Within a Blockquote (Core Markdown)

````
> ## This is an H2
>
> - This is a list
> - Inside a blockquote
>
> ```
> console.log("Code inside a blockquote");
> ```
````

Markdown Cheat Sheet

Ralph Mason, Author

Blockquote in a List Item (Core Markdown)

Indent the blockquote by 4 spaces or a tab

```
* list item:

    > Blockquote inside a list item.
```

Horizontal Rules/Line Breaks (Core Markdown)

• **
**: hit enter or end the line with two spaces
• **<hr>**: *** or --- or ___ on a line on their own

Raw HTML in Markdown

HTML elements can be used in Markdown as long as you don't indent the first/last tags in a HTML block and add a line break before/after the block.

Escaping Markdown Characters

The following Markdown characters can appear literally when escaped with a backslash (e.g. `\#`):

\\ \` * _ \{} \[] \() \# \+ \- \. \!

Links (Core Markdown)

```
[Linked text](https://sitepoint.com “Optional title")
```

```
[Reference][id]

⋮

[id]: https://sitepoint.com “Optional title"
```

```
Visit <https://sitepoint.com> for more.
```

• **[id]** can be any random identifier
• **[id]: reference** can sit on its own line, absolutely anywhere in the document
• Markdown auto-obfuscates email addresses

Images (Core Markdown)

```
![Alt text](path/to/img.jpg “Optional title")
```

```
![Reference alt text][id]

⋮

[id]: path/to/image  "Optional title"
```

```
Visit <https://sitepoint.com> for more.
```

• Links can be absolute (with http://) or relative (link to an image within your file structure)

Markdown Cheat Sheet

Ralph Mason, Author

Extended Markdown

Most Markdown processors support an extended syntax, although syntaxes vary across editors.

Popular flavors include:

Github Flavored Markdown: https://goo.gl/Pycq8Z
Markdown Extra: https://goo.gl/7f9TuE

Some widely supported elements follow:

Footnotes (Extended Markdown)

```
This line ends with a footnote reference.[^id]

[^id]: Place this footnote anywhere.
```

Strikethrough Text (Extended Markdown)

```
Let's ~~draw a line though this~~ now
```

Syntax Highlighting (Extended Markdown)

````
```javascript
  console.log("SitePoint rocks");
```
````

Tables (Extended Markdown)

Creating a table with headers:

```
Header One     | Header Two
---------------|---------------
Content Cell   | Content Cell
Content Cell   | Content Cell
```

Notes:

- There must be a header row
- There must be a separator line after the header
- There must be at least one pipe per row
- Extra pipes at the start/end of rows are optional
- MD for inline elements is allowed within cells

Set column alignment by adding colons to the separator row:

```
:--------------|---------------:|:---------------:
Left-aligned   | Right-aligned  |    Centered
```

Notes:

- Headers are centered by default
- Columns are left-aligned by default
- Colon alignment affects both header and column

About the Author

Ralph Mason is SitePoint's Web channel editor, administrator of SitePoint's magnificent web forums, and a freelance editor and web designer at Page Affairs.