Groovy Scripting for the JVM

START HERE

`groovy <file>.groovy`
Run a script

`groovy -n -e '<code>'`
Run per input line

`groovy -e '<code>'`
Run one-liner

`groovy -cp <path> <file>`
Add classpath

`groovysh`
Interactive shell

`groovy -Dkey=value <file>`
Set property

`groovy -v`
Show versions

`@Grab('g:m:v')`
Fetch a library

Script .groovy source → `compiled` → **Bytecode** in memory → `runs on` → **JVM** needs a JDK → `@Grab` → **Libraries** Maven repos

CONCEPTS

A script needs no class or main method: top-level statements just run, and args holds the arguments.

Groovy is mostly Java-compatible: semicolons, parentheses and return are optional, and public is the default.

Beware: -c sets the source encoding, it does not compile. Use groovyc to compile.

RUNNING

`groovy <file>.groovy [args]`
Run a script
Arguments arrive in the args array.

`groovy -e '<code>'`
Run an inline script

`groovy -v`
Show Groovy and JVM versions

`groovy -cp <path> <file>`
Set the classpath
Must be the first argument.

`groovy -Dkey=value <file>`
Define a system property

`groovy -d <file>`
Print full stack traces

`groovy -l <port>`
Process each line sent to a port
Default port is 1960.

`groovy -b <class> <file>`
Use a custom base script class

`groovy -pa <file>`
Keep method parameter names

`groovy --help`
List all options

ONE-LINERS

`groovy -n -e '<code>' <file>`
Run code per line, as line

`groovy -p -e '<code>' <file>`
Run per line and print result
Reassign line to change what is printed.

`groovy -a -n -e '<code>' <file>`
Split each line into split
Default separator is whitespace; -a takes a pattern.

ONE-LINERS (cont.)

`groovy -i .bak -p -e '<code>' <file>`
Edit files in place with backup
Without an extension no backup is kept.

`groovy -c <charset> <file>`
Set source encoding

SHELL

`groovysh`
Start the interactive shell

`groovysh -q`
Start without the banner

`groovysh -e '<code>'`
Evaluate code at start

`/help`
List shell commands

`/show`
List shared variables

`/load <file>`
Load a file into the buffer

`/save <file>`
Save the buffer to a file

`/history`
Show and recall history

`/inspect <expr>`
Browse an object

`/grab <g:m:v>`
Add a dependency to the shell

`/exit`
Leave the shell

DEPENDENCIES

`@Grab('<g>:<m>:<v>')`
Pull in a library
Resolved from Maven Central and cached under ~/.groovy.

`@GrabResolver(name='<n>', root='<url>')`
Add a repository

`@GrabExclude('<group>:<module>')`
Skip a transitive dependency

`@GrabConfig(systemClassLoader=true)`
Use the system class loader
Needed for JDBC drivers.

`grape install <group> <module> [<version>]`
Download into the cache

`grape list`
List cached modules

SYNTAX

`def x = 1`
Declare a variable
def is the same as Object.

SYNTAX (cont.)

`println "Hi ${name}"`
Interpolate in a GString
Only double-quoted strings interpolate.

`'''line1 line2'''`
Multi-line string
Triple double quotes interpolate too.

`/a\d+/`
Slashy string for regex

`def f = { a, b → a + b }`
Closure

`list.collect { it * 2 }`
Map over a collection
it is the implicit single parameter.

`a?.b`
Safe navigation

`a ?: b`
Elvis: default if falsy

`list*.name`
Spread: property of each item

`a <=> b`
Compare: -1, 0 or 1

`a == b`
Value equality
Use is for identity.

`obj.&method`
Method pointer

`class Foo { String name }`
Class with auto getters

`@CompileStatic`
Static type checking

`@TypeChecked`
Check types, stay dynamic

COLLECTIONS

`[1, 2, 3]`
List literal

`[a: 1, b: 2]`
Map literal

`[] as Set`
Convert to another type

`(1..5)`
Inclusive range
Use 1..<5 to exclude the end.

`list.each { println it }`
Loop over items

`list.findAll { it > 1 }`
Filter

`list.find { it > 1 }`
First match

`list.inject(0) { s, x → s + x }`
Fold to one value

`list.sum()`
Add up elements

`list.groupBy { it.size() }`
Group into a map

COLLECTIONS (cont.)

`list.sort { it.name }`
Sort by key

`list.collectEntries { [(it): it.size()] }`
Build a map

`map.each { k, v → println "$k=$v" }`
Loop over a map

`list.any { it > 1 }`
Any element matches

FILES & TEXT

`new File('<path>').text`
Read a whole file

`new File('<path>').eachLine { }`
Read line by line

`new File('<path>').text = '<data>'`
Write a file

`new File('<path>') << '<data>'`
Append to a file

`new File('.').eachFileRecurse { }`
Walk a directory tree

`'echo hi'.execute().text`
Run a shell command

`str =~ /<regex>/`
Find a match

`str ==~ /<regex>/`
Match the whole string

`new groovy.json.JsonSlurper().parseText(s)`
Parse JSON

`groovy.json.JsonOutput.toJson(obj)`
Write JSON

RECIPES

Script with a dependency

```
@Grab('org.apache.commons:commons-lang3:3.14.0')
import org.apache.commons.lang3.StringUtils
println StringUtils.capitalize('groovy')
```

Print lines that match

```
groovy -n -e 'if (line =~ /ERROR/) println line' app.log
```

Replace text in place

```
groovy -i .bak -p -e 'line.replace("foo", "bar")' *.txt
```

Fetch and parse JSON

```
def url = 'https://api.example.com/items'
def data = new groovy.json.JsonSlurper().parse(url.toURL())
data.each { println it.name }
```