Git Cheat Sheet

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

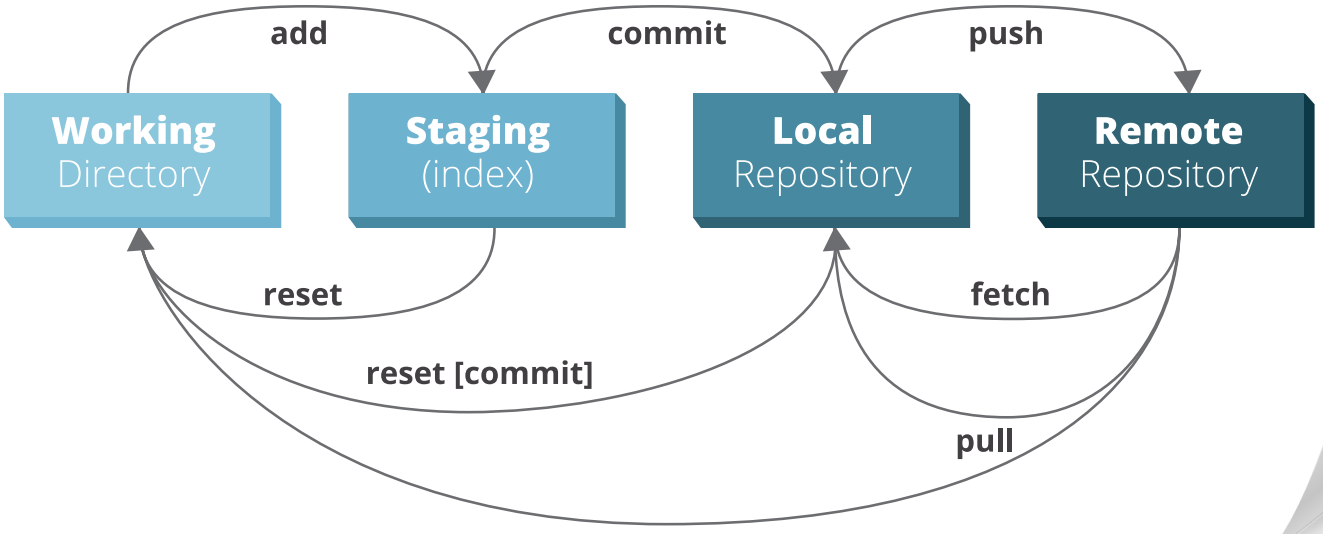
```
$ git push
```

Finally!

When in doubt, use git help

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

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



SQL cheat sheet

For more awesome cheat sheets visit rebellabs.org!

REBELLABS by ZEROTURNAROUND

Basic Queries

```
-- filter your columns
   SELECT col1, col2, col3, ... FROM table1
-- filter the rows
   WHERE col4 = 1 AND col5 = 2
-- aggregate the data
   GROUP by ...
-- limit aggregated data
   HAVING count(*) > 1
-- order of the results
   ORDER BY col2
```

Useful keywords for **SELECTS**:

DISTINCT - return unique results
BETWEEN a **AND** b - limit the range, the values can be numbers, text, or dates
LIKE - pattern search within the column text
IN (a, b, c) - check if the value is contained among given.

Data Modification

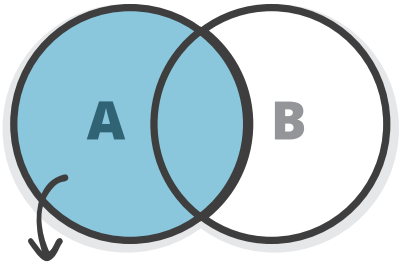
```
-- update specific data with the WHERE clause
   UPDATE table1 SET col1 = 1 WHERE col2 = 2
-- insert values manually
   INSERT INTO table1 (ID, FIRST_NAME, LAST_NAME)
      VALUES (1, 'Rebel', 'Labs');
-- or by using the results of a query
   INSERT INTO table1 (ID, FIRST_NAME, LAST_NAME)
      SELECT id, last_name, first_name FROM table2
```

Views

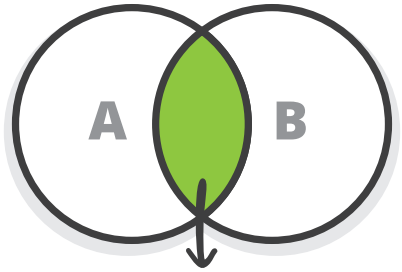
A **VIEW** is a virtual table, which is a result of a query.
They can be used to create virtual tables of complex queries.

```
CREATE VIEW view1 AS
SELECT col1, col2
FROM table1
WHERE ...
```

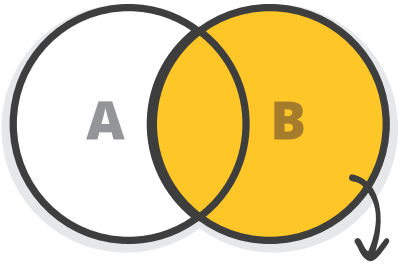
The Joy of JOINs



LEFT OUTER JOIN - *all rows from table A, even if they do not exist in table B*



INNER JOIN - *fetch the results that exist in both tables*



RIGHT OUTER JOIN - *all rows from table B, even if they do not exist in table A*

Updates on JOINed Queries

You can use **JOIN**s in your **UPDATE**s

```
UPDATE t1 SET a = 1
FROM table1 t1 JOIN table2 t2 ON t1.id = t2.t1_id
WHERE t1.col1 = 0 AND t2.col2 IS NULL;
```

NB! Use database specific syntax, it might be faster!

Semi JOINs

You can use subqueries instead of **JOIN**s:

```
SELECT col1, col2 FROM table1 WHERE id IN
   (SELECT t1_id FROM table2 WHERE date >
      CURRENT_TIMESTAMP)
```

Indexes

If you query by a column, index it!

```
CREATE INDEX index1 ON table1 (col1)
```

Don't forget:

Avoid overlapping indexes
Avoid indexing on too many columns
Indexes can speed up **DELETE** and **UPDATE** operations

Useful Utility Functions

```
-- convert strings to dates:
   TO_DATE (Oracle, PostgreSQL), STR_TO_DATE (MySQL)
-- return the first non-NULL argument:
   COALESCE (col1, col2, "default value")
-- return current time:
   CURRENT_TIMESTAMP
-- compute set operations on two result sets
   SELECT col1, col2 FROM table1
   UNION / EXCEPT / INTERSECT
   SELECT col3, col4 FROM table2;
```

Union - returns data from both queries
Except - rows from the first query that are not present in the second query
Intersect - rows that are returned from both queries

Reporting

Use aggregation functions

COUNT - return the number of rows
SUM - cumulate the values
AVG - return the average for the group
MIN / MAX - smallest / largest value

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**.