

Command Structure & Parameters

Every hashcat command follows the same pattern: the binary, the flags, and the arguments. Master the structure once and every attack type becomes a variation of the same template.

Learning Objectives

- Understand the **general syntax** of a hashcat command
 - Know the **key parameters**: `-m`, `-a`, `-o`, `-r`, `-w`, `-D`
 - Distinguish between **flags** (options) and **positional arguments** (hash file, dictionary, mask)
 - Build complete commands for **dictionary, mask, and hybrid attacks**
 - Use **sessions, incremental mode, and workload tuning** for long-running jobs
-

The Basic Command

The hashcat binary name depends on your platform, but the structure is always the same.

Linux / macOS

```
hashcat [options] hash_file [dict|mask]
```

Or explicitly: `./hashcat.bin`

Windows

```
hashcat.exe [options] hash_file [dict|mask]
```

Anatomy of a Command

```
hashcat -m 0 -a 0 -o results.txt hashes.txt rockyou.txt
#      |      |      |      |      |      |
#      |      |      |      |      |      | dictionary (argument)
#      |      |      |      |      |      | hash file (argument)
#      |      |      |      |      |      | output file (flag + value)
#      |      |      |      |      |      | attack mode (flag + value)
#      |      |      |      |      |      | hash type (flag + value)
```

Notes: Emphasize that order of flags does not matter, but the hash file must always come before the dictionary/mask arguments.

Hash Type: `-m`

The `-m` flag tells hashcat what hash algorithm to crack. Choosing the wrong mode means zero results.

Value	Algorithm	Example Use Case
0	MD5	Web app databases, legacy systems
100	SHA-1	Older web applications
1000	NTLM	Windows password hashes
1800	SHA-512 (Unix)	Linux <code>/etc/shadow</code>
3200	bcrypt	Modern web apps
22000	WPA-PBKDF2-PMKID+EAPOL	Wi-Fi auditing

Full list: `hashcat --help` or `hashcat --example-hashes`

Notes: Run `hashcat --example-hashes | head -60` live to show how many modes exist (~350+). Students should bookmark the wiki page.

Attack Mode: `-a`

The `-a` flag selects the attack strategy. Each mode determines what arguments follow the hash file.

Value	Mode	Arguments	Description
0	Dictionary	wordlist	Tries every word in a wordlist
1	Combinator	wordlist1 wordlist2	Concatenates words from two lists
3	Mask / Brute-force	mask pattern	Generates candidates from a character mask
6	Hybrid: dict + mask	wordlist mask	Appends mask patterns to dictionary words
7	Hybrid: mask + dict	mask wordlist	Prepends mask patterns to dictionary words

Key insight: The attack mode dictates the arguments. `-a 0` needs a wordlist. `-a 3` needs a mask. `-a 6` needs both. Get these wrong and hashcat will throw an error.

Output & Rules

Control where results go and how candidates are mutated.

Output: `-o` & `--outfile-format`

- `-o results.txt` — save cracked hashes to file
- `--outfile-format 1` — hash only
- `--outfile-format 2` — plain password only
- `--outfile-format 3` — hash:password (default)
- `--outfile-format 5` — hash:password:hex

Rules: `-r`

- `-r rules/best64.rule` — apply rule file
- Mutates dictionary words (capitalize, append digits, leet-speak, etc.)
- Multiple `-r` flags = chained rules
- Dramatically expands effective wordlist size

```
# Dictionary attack with rules, saving hash:password pairs
hashcat -m 0 -a 0 -o cracked.txt --outfile-format 3 -r rules/best64.rule hashes.txt rockyou.txt
```

Performance & Session Control

Long-running cracks need workload tuning and the ability to pause/resume.

Workload: `-w`

- `-w 1` — Low (desktop responsive)
- `-w 2` — Default
- `-w 3` — High (recommended for dedicated rigs)
- `-w 4` — Nightmare (may freeze UI)

Device: `-D`

- `-D 1` — CPU only
- `-D 2` — GPU only

Sessions: `--session` / `--restore`

```
# Start a named session
hashcat --session mysession -m 0 ...

# Resume later
hashcat --session mysession --restore
```

Sessions save progress to disk. Essential for multi-hour cracks.

- -D 1,2 — Both
-

Incremental Mode: `-i`

When using mask attacks, incremental mode tries all lengths from `--increment-min` to `--increment-max` instead of only the exact mask length.

Without increment

```
# Only tries exactly 8-character combinations  
hashcat -m 1000 -a 3 hashes.txt ?a?a?a?a?a?a?a
```

With increment

```
# Tries 4-char, 5-char, 6-char, 7-char, then 8-char  
hashcat -m 1000 -a 3 --increment --increment-min 4 --increment-max 8 hashes.txt ?a?a?a?a?a?a?a
```

This is powerful but dramatically increases the total keyspace. Use `--increment-min` to skip trivially short passwords and save time.

Notes: Show the keyspace calculation difference: 8-char only vs 4-to-8. The numbers are eye-opening.

Putting It All Together

A full-featured command combining multiple parameters for an NTLM mask attack.

```
hashcat -m 1000 -a 3 -o results.txt --session mysession \  
--increment --increment-min 4 --increment-max 8 \  
-w 3 hashes.txt ?a?a?a?a?a?a?a
```

Breaking it down

- `-m 1000` — NTLM hashes
- `-a 3` — Mask attack
- `-o results.txt` — Output cracked hashes to file
- `--session mysession` — Named session for pause/resume
- `--increment --increment-min 4 --increment-max 8` — Try lengths 4 through 8
- `-w 3` — High-performance workload
- `hashes.txt` — Input hash file
- `?a?a?a?a?a?a?a` — Mask: all printable characters, up to 8 positions



Notes: Have students build this command parameter by parameter in the lab. Each flag should be added one at a time so they understand what each piece does.

