SAVORY MINDS

BASH SCRIPTING Modules

Phase 0 — Orientation (what, why, how)

Goal: understand what Bash is, where it runs, how to open a terminal, safety basics.

- What is Bash? (shell + scripting interpreter)
- Terminal vs shell vs shell session vs shell script
- **Filesystems:** absolute vs relative paths (/,. .., ~)
- **Basic navigation:** pwd, ls, cd
- **Safety:** don't run random curl | bash, backups

Tiny examples + inch-by-inch

- pwd — print working directory (no args)
- ls -la
    - ls = list files
    - -l = long format, -a = include hidden. files

Common mistakes

- Running scripts without reading them
- Confusing ~ vs /home/username

Exercises

- Open terminal, run pwd, ls, cd ., ls -a.
- Show full path of home (echo $HOME).

Mini project

- Write a short note: create notes.txt with echo "My notes" > notes.txt and view it.

Phase 1 — Shell essentials & tiny syntax (symbols, quotes, redirection)

Goal: master tokens and symbols: >, >>, |, &&, ||,;, \, quotes, $(...), backticks, escaping.

Topics & inch-by-inch lines

1. **Quoting**
   - Single quotes '...' — literal, no variable expansion.
     Example: echo '$HOME' prints $HOME.
   - Double quotes "..." — allow variable expansion and most escapes.
     Example: echo "Home is $HOME".
   - No quotes → word splitting & globing. Danger with spaces.
2. **Escaping**
   - \ Escapes next char: echo "A \"quote\"".
3. **Command substitution**
   - $(date) vs `date` — runs date, inserts its output.
   - Example: echo "Today is $(date +%F)".
4. **Redirection**
   - > overwrite file: echo hi > file.txt.
   - >> append: echo more >> file.txt.
   - 2> redirect stderr: cod 2> error.log.
   - &> redirect both stdout+stderr: cmd &> output.log.
   - Example: mkdir newdir > /dev/null 2>&1 — suppress messages.
5. **Pipelines and chaining**
   - | pipe stdout of left to stdin of right: ls | grep txt.
   - && run next only if previous succeeded: mkdir d && cd d.
   - || run next only if previous failed: command || echo "failed".
   - ; run sequentially regardless of success.

6. **Brace expansion & globbing**
    - {a,b} → echo {1..3} → 1 2 3
    - * ? globbing.

Common beginner mistakes

- Missing quotes around variables containing spaces.
- Using > when you meant >> and losing data.
- Forgetting spaces with [tests.

Exercises

- Create files a.txt b.txt with echo and append a line.
- Pipe ls to grep to filter .sh files.
- Redirect stderr: run a non-existing command and capture its stderr.

Mini project

- Make a one-line command that creates a dir project, creates README.md with a line, and shows its contents.

Phase 2 — Variables, expansion, and environment (inch-by-inch)

Goal: create and use variables safely; understand environment vs local variables; variable expansion nuances.

Topics & inch-by-inch

1. **Assigning**
    - NAME="Amith" — no spaces around =.
    - NUM=5 (numbers are strings in shell).
2. **Reading**
    - $NAME or ${NAME} — use braces to disambiguate: echo "${NAME}123" vs echo "$NAME123" (bad).
    - ${#NAME} length, ${NAME:1:3} substring.
3. **Default / fallback**
    - ${VAR:-default} — if VAR unset or empty -> default (does not assign).
    - ${VAR:=default} — assign default if unset.
4. **Exporting**
    - export NAME="Amith" — makes available to child processes.
    - Permanent: echo 'export NAME="Amith"' >> ~/.bashrc then source ~/.bashrc.
5. **Arrays**
    - arr=(one two "three four")
    - Access: ${arr[0]}; all: ${arr[@]}.
6. **Command substitution**
    - FILES=$(ls -1) — careful: word splitting, use arrays if you need exact items: mapfile -t arr < <(ls -1).

Common mistakes

- myvar = "x" (spaces) → command not assignment.
- Not quoting when using variable in tests: [-f $FILE] breaks if $FILE empty or contains spaces — use [-f "$FILE"].

- Expecting export to affect the parent shell — it doesn't.

Exercises

- Create a variable, use braces to combine with text.
- Use default expansion: echo ${NOTSET:-"default"}.

Mini project

- Script that builds filenames using date: backup_$(date +%F).tar.gz.

Phase 3 — Scripts: files, shebang, permissions, execution

Goal: write real scripts, make them executable, understand shebang, paths.

Topics & inch-by-inch

1. **Shebang**
    - #!/bin/bash at top — tells kernel which interpreter to use.
    - Without it, ./script may run in current shell or fail.
2. **Creating a script**
    - echo -e '#!/bin/bash\necho hi' > hello.sh
3. **Permissions**
    - chmod +x hello.sh — explain rwx, numeric modes (chmod 755), umask.
4. **Executing**
    - ./hello.sh runs script in subprocess.
    - source hello.sh or . hello.sh runs in current shell (useful for exports).
5. **PATH**
    - PATH is colon-separated directories. export PATH="$PATH:$HOME/bin".
    - Why add a directory not a file.

Common mistakes

- Forgetting chmod +x.
- Adding file to PATH instead of directory.

- Using source when you wanted a subshell (or vice versa).

Exercises

- Create script that prints "Hello $USER", make it executable, run it.
- Add ~/bin to PATH and place script there.

Mini project

- Create hello-world script in ~/bin and run it from anywhere.

Phase 4 — Conditionals (if / elif / else), test operators

Goal: master [ ], [[ ]], test operators (-f, -d, -e, -z, -n, -s, -r, -w, -x, numeric ops -lt -gt -eq), string ops.

Topics & inch-by-inch

1. **Basic if**
    - if [condition]; then ... fi (explain spaces).
    - if [[condition]]; then ... fi (Bash extended test — safer for string ops).
2. **Numeric comparisons**
    - -lt, -le, -gt, -ge, -eq, -ne.
    - Example: if ["$a" -gt 5]; then echo more; fi
3. **String comparisons**
    - ["$a" = "$b"] or [[$a == $b]].
    - -z empty, -n nonempty.
4. **File tests**
    - -e exists, -f regular file, -d directory, -s size>0, -L symlink.
5. **Combining**
    - Logical AND/OR in test: [-f "$f"] && [-s "$f"] or [[-f $f && -s $f]].
6. **Best practices**
    - Quote variables inside tests: [-f "$f"].
    - Prefer [[ ]] for string comparisons to avoid some pitfalls.

Common mistakes

- No spaces inside [ ].
- Not quoting variables.
- Using = for numbers or -gt for strings.

Exercises

- Script that checks: file argument exists and is non-empty, else prints usage.
- Check if argument is numeric and positive.

Mini project

- Create a script safe_rm.sh that backs up a file to .bak before deleting (only if file exists and is writable).

Phase 5 — Loops (for, while, until, reading lines)

Goal: repeat tasks safely over files, lines, ranges.

Topics & inch-by-inch

1. **for loop**
    - for i in {1..5}; do echo $i; done
    - for f in *.txt; do echo "$f"; done — explain globbing and quoting.
2. **while loop**
    - while [condition]; do ...; done
    - read lines: while IFS= read -r line; do echo "$line"; done < file.txt
        - IFS= prevents trimming, -r prevents backslash escapes.
3. **until**
    - Opposite of while.
4. **break / continue**
    - break exits loop, continue skips to next iteration.

Common mistakes

- Looping with for file in $(ls *.txt) breaks on spaces — use for file in *.txt or mapfile -t.

Exercises

- Count files in directory with for loop.
- Read /etc/passwd and print usernames.

Mini project

- Build a script that rotates logs: move .log older than 7 days into archive folder.

Phase 6 — Functions, modular scripts, return codes

Goal: write reusable functions, use return codes, local variables.

Topics & inch-by-inch

1. **Define & call**
    - myfunc(){ echo "hi $1"; } then myfunc buddy.
    - Positional params inside functions: $1, $2.
2. **Return values**
    - Use return for integer exit code (0 = success). Use echo for string output and capture val=$(myfunc arg).
3. **Local variables**
    - local x=5 inside functions to avoid polluting global namespace.
4. **Sourcing libraries**
    - source lib.sh to reuse functions.

Common mistakes

- Using return with strings (it expects integer 0-255).
- Not quoting captured output.

Exercises

- Write function is_even() that returns success 0 if number even.

Mini project

- Make a utility log_and_run() that logs a command, runs it, and logs exit code.

Phase 7 — Positional parameters, get opts, input & I/O

Goal: make scripts accept arguments and options, parse flags.

Topics & inch-by-inch

1. **Positional parameters**
    - $0 script name, $1 first arg, $@ all args, $# count.
2. **Shift**
    - shift shift parameters left.
3. **getopts**
    - Parse short options: while getopts "ab:c" opt; do case $opt in a) ... ;; b) val=$OPTARG ;; esac; done
4. **read**
    - read -p "Name: " name — interactive prompt.
5. **Heredoc / Here-string**
    - cat <<EOF > file multiline write.

Common mistakes

- Using while getopts incorrectly (mixing positional arg locations).
- Forgetting to quote $@ when forwarding args: use "$@".

Exercises

- Script that accepts -n NAME and -a AGE and prints greeting.

Mini project

- Create a CLI utility with flags: ./tool.sh -v -o outdir -f filename.

Phase 8 — File & text processing (grep, sed, awk, cut, sort)

Goal: parse, transform, and extract data from text via command-line tools.

Topics & inch-by-inch

1. **grep**
    - grep -E, -i, -r, regex basics.
2. **awk**
    - Field processing: awk -F: '{print $1}' /etc/passwd.
3. **sed**
    - Stream edits: sed -n '1,10p', sed 's/foo/bar/g'.
4. **cut / sort / uniq**
    - cut -d: -f1, sort -u, uniq -c.
5. **xargs**
    - find . -name '*.log' -print0 | xargs -0 rm — explain NUL separation.

Common mistakes

- Using grep without quoting regex that contains shell metacharacters.
- Not using -print0/-0 pair with filenames containing newlines.

Exercises

- Print usernames from /etc/passwd sorted uniquely.
- Replace a word in a file using sed and make a backup.

Mini project

- Script that extracts IPs from logs and outputs top 10.

Phase 9 — Processes, job control, background tasks, cron

Goal: manage processes, run tasks in background, schedule recurring jobs.

Topics & inch-by-inch

1. **Background & foreground**
    - cmd & run in background
    - jobs, fg, bg, kill %1
2. **Signals**
    - kill PID, kill -9 (SIGKILL) — explain danger.
3. **ps / top / htop**
    - ps aux | grep name
4. **Cron & crontab**
    - crontab -e — schedule: 0 2 * * * /home/user/backup.sh
    - Explain environment differences in cron jobs.
5. **systemd timers** (mention as alternative)

Common mistakes

- Forgetting absolute paths in cron, environment variables difference, PATH missing.
- Using & without redirecting output -> broken pipes when session ends.

Exercises

- Schedule a cron job that writes a date to ~/cron_test.log every minute (for testing).

Mini project

- Create a backup script and schedule via cron to run nightly.

Phase 10 — Debugging, testing, linting, best practices

Goal: build reliable scripts, debug, and maintain style.

Topics & inch-by-inch

1. **Debugging**
    - bash -x script.sh — trace commands.
    - set -e stop on error, set -u treat unset var as error, set -o pipefail.
2. **Exit codes**
    - $? immediate previous exit code; exit N.
3. **shellcheck**
    - Linting tool to catch common pitfalls.
4. **Logging**
    - Send logs to files and syslog; rotate logs.
5. **Documentation**
    - Comments, usage messages (usage()), --help output.

Common mistakes

- Not using set -euo pipefail in CI/test scripts.
- Ignoring shellcheck warnings.

Exercises

- Add set -euo pipefail to a script, run with bash -x, fix issues.

Mini project

- Create a small test harness that tests your scripts with sample inputs and reports pass/fail.

Phase 11 — Packaging, distribution, and deployment

Goal: make scripts installable, usable by others.

Topics

- Place scripts in /usr/local/bin or ~/bin.
- Create Makefile or simple installer script: move files, set perms.
- Create .deb or tarball for distribution (overview).
- Versioning and changelog.

Exercises

- Write install.sh that copies tools to /usr/local/bin and sets perms.

Phase 12 — Projects & Capstone

Goal: combine everything into larger real-world scripts.

Project ideas

- Automated backup + rotation + email alert.
- File organizer (by ext/date) with dry-run option.
- Simple deploy script: pull from git, restart service, log results.
- Interactive CLI tool with subcommands and flags.
- Log monitor: parse logs, detect patterns, alert.

Deliverable checklist

- Robust parsing of args, help output, logging, error handling, tests, cron integration.

Appendix — quick reference (cheat sheets)

- **Basic operators**: -f -d -e -s -r -w -x
- **Numeric**: -lt -le -gt -ge -eq -ne
- **String**: == != < > -z -n
- **Redirections**: > >> 2> &> /dev/null
- **Chaining**: && || ; |
- **Loop skeletons**
  - for i in ...; do ...; done
  - while [cond]; do ...; done
  - until [cond]; do ...; done
- **Functions**
  - func(){ local x="$1"; echo "$x"; }
- **Best flags**
  - set -euo pipefail for strict scripts
  - bash -x for debugging

Study plan (suggested path)

1. Phase 0–2: basics & variables — practice daily for 3–7 days.
2. Phase 3–5: scripts, conditionals, loops — build many tiny scripts.
3. Phase 6–9: functions, args, text processing, jobs — do intermediate projects.
4. Phase 10–12: polishing, testing, packaging, capstone.

Suggestions / How we recommend you to learn (practical)

- **Practice-by-doing**: After every Live class revise & write a one-liner scripts taught in class.
- **Break things intentionally** (make mistakes, fix them). Practice extensively.

Hack-Forge | Powered by Savory Minds

Learn. Transform. Lead.

Why INDIA Trust Secure2K25 – Highlights That Speak Loudly!

- 🔥 **70,000+ students trained and mentored across India through Savory Minds**
- 📢 **India's first-ever student-powered testimonial platform to connect learners directly**
- ⭐ **Top-rated and most trusted program for 4+ years — consistently loved by students**
- 🛡️ **100% positive feedback from learners, mentors, and college leaders**
- 🤝 **50+ college MoUs across India — with growing national recognition**
- 💥 **Emotional breakthroughs — from confusion to confidence, from silence to success**
- 🧠 **Our students are now mentors, bug bounty hunters, developers, and changemakers**
- 🚀 **Backed by top cybersecurity professionals and industry experts**
- 💼 **Built for real careers, not just certificates — with a proven roadmap to jobs, startups, and more**

Important Notice – Beware of Fraudsters

Savory Minds does not engage in unsolicited calls, messages, or financial requests. Please read carefully:

- We do not ask for advance payments through random calls or messages
- We do not charge any application fee to fill the Secure2K25 form
- We will never ask you to share OTPs, passwords, or sensitive personal data
- We do not accept payments via PhonePe, Paytm, Google Pay, or personal accounts
- We do not discuss financial matters on WhatsApp, Instagram, or other informal channel

Hack-Forge | Powered by Savory Minds

Learn. Transform. Lead.

- All official communication will happen ONLY via our verified company email and portal access
- Only verified team members from Savory Minds will contact you, and only after you submit the official application form
- Your communication details will be used strictly within our secure onboarding system

---

Stay alert. Stay protected. Trust only verified sources!

For any concerns, contact our academic team through the official channels listed on our website/details given below.

For More Information & Assistance

Contact:

G. Nikitha
Head – Academic Counsellor
Savory Minds
+91 80744 86584

Feel free to reach out for any queries regarding the program, scholarships, or application process.