Shell

What is the Shell?

> A command-line interpreter that allows users to access an operating system's services

Example - list directory contents:

```
$ ls
```

Key Concept: Intermediary between user and OS kernel

Control path: user → terminal → shell → kernel → hardware

Common Shells by OS

OS	Default Shell
Linux	**bash**
macOS	**zsh**
Windows	**PowerShell**

Important Terminology

- **Terminal**: the program that provides the window/interface
- **Shell**: the program that interprets and executes commands
- **Command Prompt**: the text indicator showing the shell is ready for input

Why Learn Shell Commands?

1. **Automation**: Repetitive tasks can be scripted
2. **Efficiency**: Often faster than GUI for many tasks
3. **Remote Access**: Essential for managing servers
4. **Power**: Access to system features not available in GUI
5. **Universal**: Works on minimal systems without graphical interface

File System Hierarchy

```
/                       # root directory
/dir1/                  # 1st level directory
/dir1/dir2/             # 2nd level directory
/dir1/dir2/file1.md     # File
```

Path Types

- **Absolute**: starts from root e.g. `/home/user/documents`
- **Relative**: starts from current dir e.g. `./documents`

Special Paths

- `~` : user home dir
- `.` : current dir
- `..` : parent dir
- `-` : previous dir

Basic Navigation Commands

Command	Purpose	Example
`pwd`	print working directory	`pwd`
`ls`	list dir contents	`ls -al`
`cd`	change dir	`cd /home/user`

Common `ls` Options

```
ls -l    # Long format with details
ls -a    # Show hidden files (starting with .)
ls -h    # Human-readable file sizes
ls -t    # Sort by modification time
ls -r    # Reverse sort
ls -R    # Recursive listing
```

🎯

Active Learning: Navigation Practice

Try these commands (think about what each does):

1. `pwd`
2. `ls -la`
3. `cd ..`
4. `cd ~`
5. `ls -lht`

File/Directory Manipulation

Creating and Deleting

Command	Purpose	Example
`mkdir`	make dir	`mkdir -p path/to/dir`
`touch`	create empty file or update timestamp	`touch file.txt`
`rm`	remove files/directories	`rm -rf dir1/`
`rmdir`	remove empty dir	`rmdir dir2`

Copying and Moving

```
cp source.txt dest.txt          # Copy source to dest
cp -r source_dir/ dest_dir/     # Copy directory recursively
mv oldname.txt newname.txt      # Rename/move file
mv file.txt /new/location/      # Move to different directory
```

Pattern Matching with Wildcards

```
*        # Matches zero or more characters
?        # Matches exactly one character
[abc]    # Matches any one of a, b, or c
[0-9]    # Matches any digit
[^abc]   # Matches any character except a, b, or c
```

Examples:

```
ls *.txt            # All .txt files
rm file?.pdf        # Remove file1.pdf, file2.pdf, etc.
cp [A-Z]*.doc ~/    # Copy all .doc files starting with uppercase
```

Globbing: process by which shell expands wildcards into a list of matching pathnames

🎯

Active Learning: Wildcards Challenge

What files would these commands match?

1. `ls data_*.csv`
2. `rm test[1-3].txt`
3. `cp *.py backup/`
4. `ls [^0-9]*`

Viewing File Contents

Command	Purpose	Example
`cat`	Concatenate and display (small files)	`cat file.txt`
`less`	Page through large files	`less large_file.log`
`head`	Show first lines for a quick preview	`head -n 20 file.txt`
`tail`	Show last lines e.g. to see the latest log entries	`tail -f logfile.txt`

Tip: `tail -f` is great for monitoring log files in real-time!

Text Processing Commands

Command	Purpose	Example
`grep`	Search text patterns	`grep "error" logfile.txt`
`sed`	Stream editor	`sed 's/old/new/g' file.txt`
`awk`	Pattern scanning and processing	`awk '{print $1}' data.txt`
`sort`	Sort lines	`sort -n numbers.txt`
`uniq`	Report or filter unique lines	`uniq -c file.txt`
`cut`	Extract columns	`cut -d',' -f2 data.csv`
`wc`	Word, line, character count	`wc -l file.txt`

Grep and Regular Expressions

Connection: DFA/NFA/Regular-Expression equivalently specify a Regular Language

```
# Search for "error" in file
grep "error" logfile.txt

# Case-insensitive search
grep -i "error" logfile.txt

# Recursive search
grep -r "function" ./src/

# Show line numbers
grep -n "warning" logfile.txt

# Count matches
grep -c "failed" logfile.txt
```

Grep Essential Options

```
grep -i    # Case-insensitive
grep -v    # Invert match
grep -n    # Line numbers
grep -c    # Count matches
grep -r    # Recursive
grep -E    # Extended regex
grep -w    # Word match
grep -A/-B/-C # Context lines
```

Basic Regex Patterns

```
.          # Any character
*          # Zero or more of preceding
^          # Start of line
$          # End of line
[abc]      # Character class
[^abc]     # Negated class
\          # Escape special character
```

Grep Common Examples

Email Addresses:

```
grep -E "[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}" emails.txt
```

IP Addresses:

```
grep -E "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}" file.txt
```

Phone Numbers:

```
grep -E "([0-9]{3})-[0-9]{3}-[0-9]{4}" file.txt  # 555-123-4567
```

URLs:

```
grep -E "https?://[[:alnum:].-]+" file.txt
```

🎯

Active Learning: Grep Challenge

Given a file `logs.txt` with various error messages:

Write grep commands to:

1. Find all lines containing "ERROR" (case-insensitive)
2. Find lines that start with a date format MM/DD/YYYY
3. Count how many times "timeout" appears
4. Find all email addresses in the file

Redirection and Pipes

Standard Streams

- **stdin (0)**: standard input
- **stdout (1)**: standard output
- **stderr (2)**: standard error

Redirection Operators

```
>   # Redirect stdout (overwrite)
>>  # Redirect stdout (append)
<   # Redirect stdin
2>  # Redirect stderr
&>  # Redirect both stdout and stderr
```

Redirection Examples

```
# Save ls output to file
ls > file_list.txt

# Append to file
echo "text" >> log.txt

# Input from file, output to file
sort < unsorted.txt > sorted.txt

# Redirect only errors
command 2> errors.log

# Redirect everything
command &> all_output.log
```

Pipes: Combining Commands

Pipe (`|`): Send output of one command as input to another

```
# List only .txt files
ls -l | grep ".txt"

# Sort and remove duplicates
cat file.txt | sort | uniq

# Find Python processes
ps aux | grep python

# Show last 20 commands
history | tail -20

# Check specific disk usage
df -h | grep /dev/sda
```

Command Substitution

```
# Modern syntax
echo "Today is $(date)"

# Backtick syntax (older)
echo "Files: `ls | wc -l`"
```

Best Practice: Use `$(command)` syntax - it's more readable and nestable

🎯

Active Learning: Pipes Practice

What will these commands do?

1. `ls -l | wc -l`
2. `cat names.txt | sort | uniq > unique_names.txt`
3. `ps aux | grep python | wc -l`
4. `history | grep git | tail -5`

Predict the output, then test if you can!

Process Management Commands

Command	Purpose	Example
`ps`	List processes	`ps aux`
`top`	Interactive process view	`top`
`kill`	Terminate process	`kill -9 1234`
`jobs`	List background jobs	`jobs`
`bg`	Resume job in background	`bg %1`
`fg`	Bring job to foreground	`fg %1`

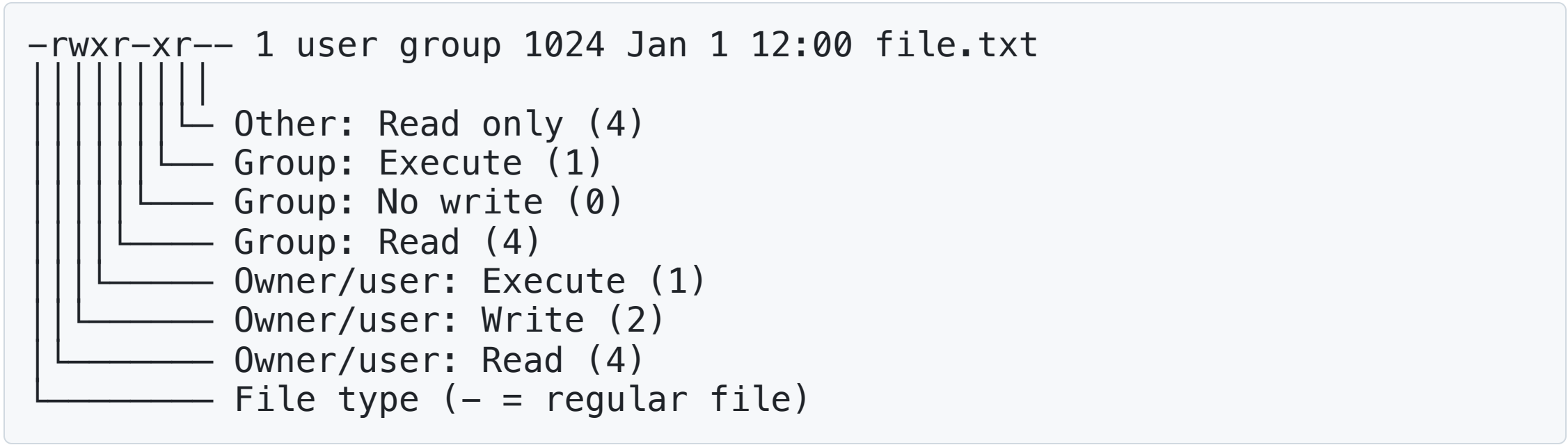
Background Execution

```
# Run in background
command &

# Suspend current process
Ctrl+Z

# Run immune to hangups
nohup command &
```

System Information

```
uname -a          # System information
df -h             # Disk usage
du -sh *          # Directory sizes
free -h           # Memory usage
uptime            # System uptime and load
```

File Permissions



Permission Commands

```
# Set permissions using octal notation
chmod 755 script.sh              # rwxr-xr-x

# Add execute permission for user
chmod u+x file.sh

# Recursive permission change
chmod -R 644 documents/

# Change ownership
chown user:group file.txt
```

Octal Notation

- Read (r) = 4
- Write (w) = 2
- Execute (x) = 1

Shell Scripting Basics

```
#!/bin/bash
# This is a comment

echo "Hello, World!"
NAME="Student"
echo "Welcome, $NAME"

# Conditionals
if [ -f "file.txt" ]; then
    echo "File exists"
fi

# Loops
for i in {1..5}; do
    echo "Number: $i"
done
```

Making Scripts Executable

```
# Make script executable
chmod +x script.sh

# Run the script
./script.sh
```

Command Line Arguments

```
#!/bin/bash
echo "Script name: $0"
echo "First argument: $1"
echo "All arguments: $@"
echo "Number of arguments: $#"
```

🎯

Active Learning: Script Writing

Create a script that:

1. Takes a filename as an argument
2. Checks if the file exists
3. If it exists, count the number of lines
4. If it doesn't, create an empty file

Safety Tips

1. Use `rm -i` for interactive deletion
2. Test with `echo` before running destructive commands
3. Use `--dry-run` when available
4. Keep backups before bulk operations

Example: Testing Before Deletion

```
# First, see what would be deleted
echo rm *.log

# Then execute if it looks right
rm *.log
```

Efficiency Tips

Shortcuts

```
# Tab completion: Press Tab to auto-complete
# History search: Ctrl+R for reverse search
```

Aliases

```
alias ll='ls -la'
alias ..='cd ..'
```

Keyboard Shortcuts

- **Ctrl+A**: Beginning of line
- **Ctrl+E**: End of line
- **Ctrl+K**: Delete to end of line

Getting Help: RTFM

```
man command         # Manual pages
command --help      # Built-in help
which command       # Find command location
type command        # Show command type
apropos keyword     # Search manual pages
```

Remember: `man` is your friend!

Summary: Key Takeaways

✓ Shell provides powerful command-line access to OS
✓ Navigate with `pwd`, `ls`, `cd`
✓ Manipulate files with `cp`, `mv`, `rm`
✓ Search and process text with `grep`, `sed`, `awk`
✓ Combine commands with pipes (`|`) and redirection (`>`)
✓ Manage processes and permissions
✓ Automate tasks with shell scripts

Practice is key! The more you use the shell, the more efficient you'll become.

Resources for Further Learning:

- Manual pages: `man <command>`
- Online tutorials: explainshell.com
- Practice: overthewire.org/wargames/bandit/