Comprehensive Revision Notes on Linux Scripting and Command Line Basics

Introduction to Bash Scripting

What is Bash Scripting?

Bash scripting is a crucial skill for automating repetitive tasks on Linux systems. It involves writing sequences of commands for the shell to execute.

Starting a Bash Script

- **Shebang**: Begin scripts with `#!/bin/bash` to denote the interpreter.
- **Variables**: Define variables using `VAR_NAME=value` with no spaces around `=`.

Basic Bash Script Structure

1. **Check Directory Exists**: Use `[ -d "$directory" ]` to test if a directory exists【4:0†source】.
2. **Count .log Files**: Use `find $directory -type f -name "*.log" | wc -l` 【4:1†source】.
3. **User Input**: `read -p "Enter backup directory name: " backup_dir` 【4:0†source】.
4. **Loop Through Files**: Utilize `for file in $directory/*.log; do ... done` 【4:9†source】.
5. **Print File Names & Errors**:
   - List files with `echo $file`.
   - Use `grep -c "error" $file` to count lines containing "error"【4:9†source】.
6. **Archiving Files**: `tar -cvzf archive-$(date +%F).tar.gz *.log` 【4:9†source】.

- **If-Else Statements**: Manage flow in scripts with `if [ condition ]; then ... else ... fi`.
- **String and Numeric Comparisons**:
  - Strings: `=` for equality, `!=` for inequality.
  - Numbers: `-eq`, `-ne`, `-lt`, `-le`, `-gt`, `-ge` for equal, not equal, less than, etc.【4:8†source】.

Command Execution and Logging

- **Redirect Outputs**: Use `exec > log_file 2>&1` to log all script output (standard and error) to a file【4:9†source】.

Command Line Utilities

Basic Commands

- `echo` : Prints output to the terminal.
- `find` : Searches for files in a directory hierarchy.
- `grep` : Searches text using patterns.
- `wc -l` : Counts lines from input.

Advanced Command Usage

- **Parameter Expansion**: Use `${VAR}` syntax to access variable content.
- **Command Substitution**: Backticks or `$()` are used to capture command output: `current_date=$(date)`【4:9†source】.

Troubleshooting Scripts

- Common issues include syntax errors and incorrect path specifications.
- Debugging involves checking script outputs and ensuring proper environment setup where necessary.

File Permissions and Conditions

- `-f` : File exists and is a regular file.
- `-d` : Exists and is a directory.
- `-r` : Readable by the user.
- `-w` : Writable by the user.
- `-x` : Executable by the user【4:15†source】.

Practical Script Writing

Example Script Structure

Below is a simplified outline of a script based on the class transcript:

```
#!/bin/bash

# Define the log directory
LOG_DIR="/var/log/myapp"

# Check if log directory exists
if [ -d "$LOG_DIR" ]; then
  echo "Log directory exists."
else
  echo "Log directory does not exist."
fi

# Count .log files
log_count=$(find $LOG_DIR -type f -name "*.log" | wc -l)
echo "Number of log files: $log_count"

# Take user input for backup directory
read -p "Enter backup directory name: " backup_dir

# Loop through log files
for file in $LOG_DIR/*.log; do
  echo "Processing file: $file"
  error_count=$(grep -c "error" "$file")
  echo "Number of errors in $file: $error_count"
done

# Archive log files
tar -cvzf backup-$backup_dir-$(date +%F).tar.gz $LOG_DIR/*.log
```

This script checks for a directory, counts .log files, takes user input, processes each file to count error lines, and archives the logs【4:0†source】【4:1†source】【4:6†source】.

Conclusion

This class covers essential aspects of bash scripting and gives a foundational understanding of command line operations prevalent in Linux environments. The ability to automate processes enhances productivity and efficiency, particularly in environments requiring frequent management of system resources and files.