Module 1: Unix Environment and Basic Commands

1.1 Introduction to Unix

Unix is a powerful, multiuser, multitasking operating system originally developed in the 1970s. Modern variants include Linux, macOS, and BSD systems.

Key Characteristics:
- Hierarchical file system
- Everything is a file (including devices)
- Case-sensitive commands and filenames
- Strong security and permission model
- Powerful command-line interface (CLI)

1.2 Unix File System Structure

The Unix file system follows a tree-like hierarchy starting from the root directory /.

Important Directories:
- / - Root directory
- /home - User home directories
- /bin - Essential command binaries
- /etc - System configuration files
- /var - Variable data (logs, temporary files)
- /tmp - Temporary files
- /usr - User programs and data

1.3 Basic Unix Commands

Navigation Commands

```
pwd                     # Print working directory
ls                      # List directory contents
ls -l                   # Long format listing
ls -a                   # Show hidden files
ls -lh                  # Human-readable file sizes
cd /path/to/directory   # Change directory
cd ..                   # Move up one directory
cd ~                    # Go to home directory
cd -                    # Go to previous directory
```

File and Directory Operations

```
mkdir dirname          # Create directory
mkdir -p dir1/dir2/dir3  # Create nested directories
rmdir dirname          # Remove empty directory
touch filename         # Create empty file or update timestamp
cp source dest         # Copy file
cp -r source dest      # Copy directory recursively
mv source dest         # Move or rename file
rm filename            # Remove file
rm -r dirname          # Remove directory and contents
rm -rf dirname         # Force remove (use carefully!)
```

File Content Commands

```
cat filename          # Display file contents
less filename         # View file with pagination
head filename         # Show first 10 lines
head -n 20 filename    # Show first 20 lines
tail filename         # Show last 10 lines
tail -f filename      # Follow file in real-time (logs)
wc filename           # Count lines, words, characters
wc -l filename        # Count lines only
```

File Permissions

```
ls -l                 # View permissions
chmod 755 filename    # Change permissions (rwxr-xr-x)
chmod +x filename     # Add execute permission
chmod -w filename     # Remove write permission
chown user:group file # Change ownership
```

Permission Numbers:
- 4 = read ®
- 2 = write (w)
- 1 = execute (x)
- 755 = rwxr-xr-x (owner: full, group/others: read+execute)
- 644 = rw-r–r-- (owner: read+write, others: read only)

Text Processing Commands

```
grep "pattern" filename      # Search for pattern in file
grep -r "pattern" directory  # Recursive search
grep -i "pattern" filename   # Case-insensitive search
find /path -name "*.txt"     # Find files by name
```

```
sed 's/old/new/g' file         # Replace text in file
awk '{print $1}' file          # Print first column
sort filename                  # Sort lines
uniq filename                  # Remove duplicate lines
```

System Information

```
whoami                  # Current username
hostname                # Computer name
uname -a                # System information
df -h                   # Disk space usage
du -sh directory        # Directory size
ps aux                  # Running processes
top                     # Real-time process monitor
kill PID                # Terminate process
```

Redirection and Pipes

```
command > file          # Redirect output to file (overwrite)
command >> file         # Append output to file
command < file          # Read input from file
command1 | command2     # Pipe output to another command
command 2> error.log    # Redirect error messages
```

Example: cat file.txt | grep "error" | sort | uniq > results.txt

---

Module 2: Version Control Systems (VCS)

2.1 What is Version Control?

Version Control Systems track changes to files over time, allowing you to recall specific versions later.
Key Benefits:
- Track history of changes
- Collaborate with multiple developers
- Revert to previous versions
- Experiment with branches safely
- Backup and recovery
- Understanding who changed what and why

2.2 Types of VCS

Local VCS

- Simple database keeping file changes on local machine
- Limited collaboration capabilities
- Example: RCS (Revision Control System)

Centralized VCS (CVCS)

- Single central server contains all versioned files
- Clients check out files from central place
- Examples: SVN, Perforce
- **Limitations:** Single point of failure, requires network access

Distributed VCS (DVCS)

- Every client has full repository mirror
- Can work offline and sync later
- No single point of failure
- Examples: Git, Mercurial

2.3 Why We Need Version Control

For Individual Developers:
- Maintain complete project history
- Experiment without breaking working code
- Document changes with commit messages
- Roll back mistakes easily

For Teams:
- Multiple people can work simultaneously
- Track who made which changes
- Merge contributions from different developers
- Code review and quality control
- Release management

Module 3: Git - Distributed Version Control

3.1 Git Basics

Installing Git

```
# Linux (Debian/Ubuntu)
sudo apt-get install git
```

```
# macOS
brew install git

# Verify installation
git --version
```

Git Configuration

```
# Set user information
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"

# View configuration
git config --list

# Set default editor
git config --global core.editor "vim"
```

3.2 Git Repository Concepts

Repository (Repo): A directory containing your project files and Git's tracking information
Working Directory: Your current file system where you edit files
Staging Area (Index): Intermediate area where you prepare commits
Local Repository: Git database on your machine (.git directory)
Remote Repository: Repository hosted on a server (GitHub, GitLab)

3.3 Creating and Initializing Repositories

```
# Initialize new repository
git init

# Clone existing repository
git clone https://github.com/username/repository.git
git clone https://github.com/username/repository.git new-folder-name

# Check repository status
git status
```

3.4 Basic Git Workflow

```
# 1. Check current status
```

```
git status

# 2. Add files to staging area
git add filename              # Add specific file
git add .                     # Add all files
git add *.js                  # Add all JS files
git add folder/               # Add entire folder

# 3. Commit changes
git commit -m "Descriptive message"
git commit -am "Message"      # Add and commit modified files

# 4. View commit history
git log
git log --oneline             # Compact view
git log --graph --oneline     # Visual branch view
git log -n 5                  # Last 5 commits
```

3.5 Committing Changes

Best Practices for Commits

- Make small, focused commits
- Write clear, descriptive commit messages
- Present tense: "Add feature" not "Added feature"
- First line: brief summary (50 chars)
- Detailed explanation after blank line if needed

Example Good Commit Messages:
```
Add user authentication feature

- Implement login endpoint
- Add password hashing with bcrypt
- Create JWT token generation
- Add middleware for protected routes
```

Viewing Changes

```
# Show unstaged changes
git diff

# Show staged changes
git diff --staged

# Show changes for specific file
```

```
git diff filename

# Compare commits
git diff commit1 commit2
```

Undoing Changes

```
# Unstage file
git reset filename
git reset HEAD filename

# Discard changes in working directory
git checkout -- filename
git restore filename         # Modern Git

# Amend last commit
git commit --amend -m "New message"

# Revert commit (creates new commit)
git revert commit-hash

# Reset to previous commit (dangerous!)
git reset --hard commit-hash
```

Module 4: Branching and Merging

4.1 Understanding Branches

Branches allow you to diverge from the main line of development and work independently
without affecting the main codebase.
Default Branch: main (formerly master)

4.2 Branch Operations

```
# List branches
git branch                       # Local branches
git branch -a                    # All branches (local + remote)

# Create new branch
git branch feature-branch

# Switch to branch
```

```
git checkout feature-branch
git switch feature-branch      # Modern Git

# Create and switch in one command
git checkout -b new-feature
git switch -c new-feature

# Delete branch
git branch -d branch-name      # Safe delete (merged only)
git branch -D branch-name      # Force delete

# Rename branch
git branch -m old-name new-name
```

4.3 Merging Branches

```
# Merge branch into current branch
git checkout main
git merge feature-branch

# Abort merge if conflicts
git merge --abort
```

Types of Merges:
Fast-Forward Merge: When target branch hasn't changed since branch creation
```
main:      A---B
                \
feature:         C---D

# After merge
main:      A---B---C---D
```

Three-Way Merge: When both branches have new commits
```
main:      A---B---E
                \
feature:         C---D

# After merge
main:      A---B---E---F (merge commit)
                \     /
feature:         C---D
```

4.4 Resolving Conflicts

Conflicts occur when the same part of a file is modified in both branches.
Conflict Resolution Process:
1. Git marks conflicts in files:

```
<<<<<<< HEAD
Current branch content
=======
Incoming branch content
>>>>>>> feature-branch
```

2. Manually edit files to resolve conflicts
3. Remove conflict markers
4. Stage resolved files:

```
git add resolved-file
```

5. Complete the merge:

```
git commit -m "Resolve merge conflict"
```

Tools for Conflict Resolution:

```
# Use merge tool
git mergetool

# View files with conflicts
git status

# Abort merge
git merge --abort
```

4.5 Rebasing (Advanced)

Rebasing rewrites commit history by moving commits to a new base.

```
# Rebase current branch onto main
git checkout feature-branch
git rebase main

# Interactive rebase (edit, squash commits)
git rebase -i HEAD~3
```

Warning: Never rebase commits that have been pushed to shared repositories.

Module 5: Working with Remote Repositories

5.1 Remote Repository Basics

```
# View remote repositories
git remote -v

# Add remote repository
git remote add origin https://github.com/username/repo.git

# Change remote URL
git remote set-url origin new-url

# Remove remote
git remote remove origin
```

5.2 Pushing and Pulling

```
# Push to remote
git push origin main
git push origin feature-branch

# Push and set upstream
git push -u origin main

# Force push (dangerous!)
git push --force origin main

# Pull from remote (fetch + merge)
git pull origin main

# Fetch without merging
git fetch origin

# Pull with rebase
git pull --rebase origin main
```

5.3 Tracking Remote Branches

```
# List remote branches
```

```
git branch -r

# Create local branch from remote
git checkout -b local-branch origin/remote-branch
git checkout --track origin/remote-branch

# Delete remote branch
git push origin --delete branch-name
```

Module 6: GitHub

6.1 Introduction to GitHub

GitHub is a web-based platform for hosting Git repositories with additional collaboration features.
Key Features:
- Repository hosting
- Issue tracking
- Pull requests
- Project management
- Actions (CI/CD)
- GitHub Pages (static hosting)
- Wikis and documentation

6.2 Creating a GitHub Repository

Via Web Interface:
1. Click "New Repository"
2. Enter repository name
3. Choose public or private
4. Initialize with README (optional)
5. Add .gitignore template (optional)
6. Choose license (optional)

Connecting Local Repository to GitHub:
```
# Create local repository
git init
git add .
git commit -m "Initial commit"

# Connect to GitHub
git remote add origin https://github.com/username/repo.git
git push -u origin main
```

6.3 GitHub Workflow

Forking: Create your own copy of someone else's repository
Cloning: Download repository to local machine
Pull Request (PR): Propose changes to a repository
Typical Workflow:
1. Fork repository
2. Clone to local machine
3. Create feature branch
4. Make changes and commit
5. Push to your fork
6. Create pull request
7. Code review and discussion
8. Merge pull request

6.4 Pull Requests

```
# Create feature branch
git checkout -b fix-bug

# Make changes and commit
git add .
git commit -m "Fix login bug"

# Push to GitHub
git push origin fix-bug
```

Then on GitHub:
1. Click "New Pull Request"
2. Select branches to compare
3. Add title and description
4. Assign reviewers
5. Submit pull request

6.5 Issues and Project Management

Issues: Track bugs, enhancements, and tasks
- Use labels (bug, enhancement, help wanted)
- Assign to team members
- Reference in commits: git commit -m "Fix #42: Login error"

Projects: Kanban-style boards for task management
Milestones: Group issues for releases or sprints

6.6 GitHub Best Practices

README.md:
- Project description
- Installation instructions
- Usage examples
- Contributing guidelines
- License information

.gitignore:
```
# Node.js
node_modules/
npm-debug.log

# Python
__pycache__/
*.pyc
venv/

# IDE
.vscode/
.idea/

# OS
.DS_Store
Thumbs.db
```

Branch Protection Rules:
- Require pull request reviews
- Require status checks
- Prevent force pushes
- Require signed commits

6.7 GitHub Actions (CI/CD)

Automate workflows with GitHub Actions:
```
# .github/workflows/test.yml
name: Run Tests

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Run tests
        run: npm test
```

Module 7: Advanced Git Concepts

7.1 Git Tags

```
# Create lightweight tag
git tag v1.0.0

# Create annotated tag
git tag -a v1.0.0 -m "Release version 1.0.0"

# List tags
git tag

# Push tags to remote
git push origin v1.0.0
git push origin --tags

# Delete tag
git tag -d v1.0.0
git push origin --delete v1.0.0
```

7.2 Git Stash

Temporarily save uncommitted changes:
```
# Stash changes
git stash
git stash save "Work in progress"

# List stashes
git stash list

# Apply stash
git stash apply
git stash apply stash@{0}

# Apply and remove stash
git stash pop

# Delete stash
git stash drop stash@{0}

# Clear all stashes
```

```
git stash clear
```

7.3 Cherry-Picking

Apply specific commits from one branch to another:
```
git cherry-pick commit-hash
git cherry-pick commit1 commit2
```

7.4 Git Submodules

Include other Git repositories as subdirectories:
```
# Add submodule
git submodule add https://github.com/user/repo.git path/to/submodule

# Clone repository with submodules
git clone --recursive repository-url

# Update submodules
git submodule update --init --recursive
```

Module 8: Practical Exercises

Exercise 1: Basic Git Workflow

1. Create a new directory and initialize Git
2. Create a README.md file
3. Add and commit the file
4. Make changes and commit again
5. View commit history

Exercise 2: Branching and Merging

1. Create a new branch called "feature"
2. Add new files in the feature branch
3. Switch back to main branch
4. Merge feature branch into main
5. Delete the feature branch

Exercise 3: Conflict Resolution

1. Create two branches from main
2. Modify the same line in a file in both branches
3. Merge one branch into main
4. Attempt to merge the second branch
5. Resolve the conflict manually

Exercise 4: GitHub Collaboration

1. Fork a sample repository on GitHub
2. Clone it to your local machine
3. Create a new branch
4. Make changes and push to your fork
5. Create a pull request to the original repository

Exercise 5: Advanced Workflow

1. Initialize a repository with multiple files
2. Create a .gitignore file
3. Use git stash to save work in progress
4. Create and tag a release
5. Push everything to GitHub including tags

Common Git Commands Reference

Essential Commands

```
git init                    # Initialize repository
git clone <url>             # Clone repository
git status                  # Check status
git add <file>              # Stage file
git commit -m "message"     # Commit changes
git push                    # Push to remote
git pull                    # Pull from remote
git branch                  # List branches
git checkout <branch>       # Switch branch
git merge <branch>          # Merge branch
git log                     # View history
```

Troubleshooting Commands

```
git status                  # Check current state
git log --oneline           # Quick history view
git diff                    # See changes
git reset HEAD <file>       # Unstage file
git checkout -- <file>      # Discard changes
git clean -n                # See untracked files
git clean -fd               # Remove untracked files
git reflog                  # View all reference logs
```

Tips and Best Practices

1. **Commit Often:** Make small, frequent commits rather than large ones
2. **Write Clear Messages:** Future you will thank present you
3. **Use Branches:** Keep main stable, experiment in branches
4. **Pull Before Push:** Always sync with remote before pushing
5. **Review Before Commit:** Use git status and git diff
6. **Backup Important Work:** Push to remote regularly
7. **Learn to Read Documentation:** git help <command>
8. **Practice in Test Repository:** Experiment safely
9. **Use .gitignore:** Don't commit unnecessary files
10. **Collaborate Effectively:** Use pull requests and code reviews

Resources for Further Learning

- **Official Git Documentation:** https://git-scm.com/doc
- **GitHub Guides:** https://guides.github.com
- **Interactive Git Learning:** https://learngitbranching.js.org
- **Git Cheat Sheet:** https://education.github.com/git-cheat-sheet-education.pdf
- **Pro Git Book (Free):** https://git-scm.com/book/en/v2