Single Drive Backup System for Research Projects

Overview and Purpose

This comprehensive backup system is designed specifically for researchers managing large project directories with hundreds of individual Git repositories. The system transforms a complex data protection challenge into an automated, reliable solution that runs quietly in the background while you focus on your research.

The Challenge: Managing 20GB+ of research data across 300+ individual project repositories requires a backup strategy that handles both the scale and the unique nature of academic work - where each project might have different collaborators, publication timelines, and data sensitivity requirements.

The Solution: A multi-layered backup system using a single external drive combined with cloud storage, providing multiple recovery points from recent file changes (hourly snapshots) to long-term archives (monthly compressed backups), while maintaining individual Git repository integrity.

Why This Architecture Works

The Research Workflow Reality

Academic research isn't like typical software development. You might:

- Work on 5-10 active projects simultaneously
- Have dormant projects that suddenly become active when reviewers request changes
- Need to access specific versions of analysis code from months ago
- Collaborate with different people on different projects
- Have projects at various stages from early exploration to published papers

This system accommodates these realities by maintaining individual Git repositories for each project while providing unified backup infrastructure.

Storage Strategy Explained

Single Drive (1TB) Partition Layout:

- **700GB: Time Machine** - Complete system backup providing full recovery capability
- **200GB: Project snapshots** - Frequent incremental backups using space-efficient hard links
- **100GB: Archives & mirror** - Compressed long-term storage and daily project mirrors

This layout prioritizes system-wide recovery (Time Machine gets the most space) while ensuring frequent, granular backup of your research projects.

The 3-2-1 Rule Implementation

- **3 copies of data**: Original + Time Machine + Cloud storage
- **2 different storage types**: Local external drive + cloud services
- **1 offsite backup**: Cloud storage (Google Drive primary, Dropbox secondary)

Understanding the Backup Layers

Layer 1: Real-Time Protection (Git Repositories)

Every project directory contains its own Git repository, automatically committing and pushing changes as you work. This provides:

- **Immediate protection**: Changes are committed locally within 5 minutes of saving
- **Version history**: Complete history of every file in every project
- **Collaboration sync**: Automatic pushes to GitHub keep collaborators current
- **Granular recovery**: Restore individual files or entire project states

Layer 2: Hourly Snapshots (Hard-Linked)

Every hour, the system creates a complete snapshot of your entire project directory using hard links. This provides:

- **Space efficiency**: Only changed files use additional storage space
- **Fast recovery**: Browse snapshots like regular directories
- **Frequent capture**: Minimal work loss (maximum 1 hour)
- **Multiple recovery points**: Last 24 hours of work accessible instantly

Layer 3: Daily Mirror (Complete Copy)

Once per day, the system creates a complete synchronized copy of your projects. This provides:

- **Full redundancy**: Exact copy independent of original
- **Cross-platform compatibility**: Standard file structure readable anywhere
- **Offline access**: Complete project access even without internet
- **Bulk operations**: Easy to copy entire research portfolio

Layer 4: Weekly/Monthly Archives (Compressed)

Regular compressed archives provide long-term storage and portability: - **Space efficiency**: Significant compression for long-term storage - **Portability**: Single file contains entire research portfolio - **Long-term preservation**: Monthly

archives kept for 6 months - **Disaster recovery**: Complete research recovery from single archive

Layer 5: Cloud Synchronization (Offsite)

Continuous cloud synchronization ensures offsite protection: - **Geographic diversity**: Data stored in multiple data centers - **Accessibility**: Access from any device, anywhere - **Sharing capability**: Easy to share specific projects with collaborators - **Redundancy**: Two cloud providers (Google Drive + Dropbox) for extra protection

Drive Setup and Partitioning

Understanding the Partitioning Strategy

The single drive is divided into three partitions, each optimized for its specific role:

```
# Single drive: Time Machine + Project snapshots + Archives

diskutil partitionDisk /dev/diskX JHFS+ "TimeMachine" 700G JHFS+ \
"PrjSnapshots" 200G JHFS+ "PrjArchive" 100G
```

Why these sizes?

- **700GB for Time Machine**: Provides several months of complete system backups, allowing recovery from major system failures
- **200GB for snapshots**: Sufficient for 24+ hours of hourly snapshots with hard-linking efficiency
- **100GB for archives**: Holds 4+ weeks of compressed weekly backups plus monthly archives

What is hard-linking? Hard links allow multiple directory entries to point to the same physical file data. If a file doesn't change between snapshots, both snapshots point to the same data on disk, using no additional space. Only modified files consume additional storage.

Directory Structure Purpose

```
# Create organized directory structure
mkdir -p /Volumes/PrjSnapshots/hourly
mkdir -p /Volumes/PrjArchive/weekly
mkdir -p /Volumes/PrjArchive/monthly
mkdir -p /Volumes/PrjArchive/current_mirror
```

Each directory serves a specific temporal role: - **hourly/**: Fast-access recent snapshots for immediate recovery - **weekly/**: Compressed archives for medium-

term storage - **monthly/**: Long-term preservation archives - **current_mirror/**: Always-current complete copy for bulk operations

The Core Scripts Explained

The backup system consists of several specialized scripts, each handling a specific aspect of data protection. Here's how they work together:

1. Master Backup Script (`prj_backup_single.sh`)

Purpose: Orchestrates all backup operations in the correct sequence **When it runs**: Every hour via launchd **What it does**: Coordinates Git sync, snapshot creation, mirror synchronization, and cloud backup

2. Space Management Script (`space_manager.sh`)

Purpose: Monitors disk usage and automatically cleans old backups when space runs low **When it runs**: Before each backup operation **What it does**: Prevents disk full scenarios by intelligently removing oldest backups

3. Status Monitoring Script (`backup_status_single.sh`)

Purpose: Provides comprehensive overview of backup system health **When it runs**: On-demand when you need to check system status **What it does**: Reports on all backup layers, space usage, and recent operations

4. Recovery Helper Script (`recovery_helper_single.sh`)

Purpose: Guides you through available recovery options during data loss scenarios **When it runs**: On-demand when you need to restore data **What it does**: Lists all available backup sources and provides exact recovery commands

5. Weekly Archive Script (`weekly_archive_single.sh`)

Purpose: Creates compressed long-term archives of your entire research portfolio **When it runs**: Weekly via cron (Sunday 2 AM) **What it does**: Creates space-efficient compressed archives with intelligent cleanup

How the Scripts Work Together

Typical Hourly Cycle (Automated)

1. **Git Sync**: Commits and pushes changes across all 300 repositories
2. **Space Check**: Verifies sufficient space for new snapshot
3. **Snapshot Creation**: Creates hard-linked incremental snapshot
4. **Mirror Sync**: Syncs daily mirror (if it's been 24+ hours)
5. **Cloud Sync**: Synchronizes with Google Drive
6. **Cleanup**: Removes snapshots older than 24 hours

Weekly Cycle (Sunday)

1. **Archive Creation**: Compresses entire project directory
2. **Monthly Check**: Creates monthly archive if first Sunday of month
3. **Space Management**: Removes old archives to prevent disk full
4. **Secondary Cloud**: Syncs with Dropbox for additional redundancy

Your Daily Interaction

The system is designed to require minimal daily interaction: - **Morning**: Quick status check (30 seconds) - **During work**: Normal editing and saving (automatic backups) - **Evening**: Optional status verification

Vim Integration Philosophy

The Vim integration is designed around the principle that your editor should be aware of your backup system, not the other way around. Commands are organized by frequency of use:

Daily commands (quick mappings): - `<leader>bs` - Backup status (most frequent) - `<leader>gs` - Git sync current project - `<leader>gd` - Git dashboard (all projects)

Weekly commands (full names): - `:RepoHealth` - Check repository integrity - `:SpaceCheck` - Verify disk space - `:BackupNow` - Force immediate backup

Backup Schedule Explained

Frequency	Action	Purpose	Storage Location
Hourly	Incremental snapshots	Protect against accidental changes	PrjSnapshots
Hourly	Cloud sync	Offsite protection	Google Drive
Daily	Mirror sync	Complete redundant copy	PrjArchive
Daily	Git operations	Version control sync	GitHub (300 repos)
Weekly	Compressed archives	Long-term storage	PrjArchive
Weekly	Secondary cloud	Additional redundancy	Dropbox
Monthly	Long-term archives	Preservation	PrjArchive

Recovery Scenarios and Options

Scenario 1: "I accidentally deleted a file 2 hours ago"

Solution: Hourly snapshots **Recovery time**: 2 minutes **Command**: Browse `/Volumes/PrjSnapshots/hourly/snapshot_TIMESTAMP/`

Scenario 2: "I need to revert a project to last week's state"

Solution: Git repository history or weekly archive **Recovery time**: 5 minutes **Command**: `git log` + `git reset` or extract from weekly archive

Scenario 3: "My laptop was stolen"

Solution: Cloud backup + Git repositories **Recovery time**: 1-2 hours (depending on internet speed) **Command**: Clone from GitHub + download from Google Drive

Scenario 4: "My external drive failed"

Solution: Cloud backup + Git repositories **Recovery time**: 2-4 hours (full restoration) **Command**: Bulk clone from GitHub + restore from cloud

Scenario 5: "I need to access a project from 3 months ago"

Solution: Monthly archives + Git repository **Recovery time**: 10 minutes **Command**: Extract from monthly archive or clone specific commit

Space Efficiency Explained

Hard-Link Efficiency

Traditional backups create complete copies of all files, consuming massive storage. Hard-linked snapshots share unchanged files between snapshots:

Example with 10 hourly snapshots: - Traditional: $10 \times 20\text{GB} = 200\text{GB}$ used - Hard-linked: ~25GB used (assuming 20% daily change rate)

Compression Efficiency

Weekly and monthly archives use gzip compression, typically achieving: - Source code: 80-90% compression - Documentation: 60-70% compression - Data files: 30-50% compression (varies by type)

Cleanup Strategy

The system implements intelligent cleanup: 1. **Immediate cleanup**: Remove snapshots older than 24 hours 2. **Space-based cleanup**: More aggressive cleanup when space runs low 3. **Prioritized retention**: Keep newest snapshots, oldest monthly archives

Implementation Steps

Phase 1: Basic Setup (30 minutes)

1. Partition your external drive
2. Install and configure the core backup script
3. Set up Time Machine
4. Test manual backup execution

Phase 2: Automation (15 minutes)

1. Install launchd configuration
2. Set up weekly cron job
3. Configure cloud synchronization
4. Test automated execution

Phase 3: Integration (15 minutes)

1. Add Vim commands to your .vimrc
2. Set up Git repository discovery
3. Configure monitoring scripts
4. Test recovery procedures

Phase 4: Optimization (ongoing)

1. Monitor space usage patterns
2. Adjust cleanup schedules based on your workflow
3. Fine-tune cloud sync settings
4. Customize scripts for your specific needs

Monitoring and Maintenance

Daily Monitoring (2 minutes)

- Check backup status via Vim: `:BackupStatus`
- Verify no error messages in recent logs
- Confirm external drive is connected and healthy

Weekly Maintenance (5 minutes)

- Review space usage across all partitions
- Check repository health: `:RepoHealth`
- Verify cloud synchronization is current
- Test recovery of a sample file

Monthly Review (15 minutes)

- Verify monthly archives are being created
- Test complete project recovery from archive

- Review and update exclude patterns if needed
- Check for any repositories that need attention

Troubleshooting Common Issues

"Backup script fails with permission errors"

Cause: Script doesn't have necessary file permissions **Solution**: Ensure scripts are executable: `chmod +x ~/Scripts/*.sh`

"Snapshots using too much space"

Cause: Hard links not working properly **Solution**: Verify rsync supports hard links and file system is HFS+/APFS

"Git operations failing for some repositories"

Cause: Remote repositories moved or authentication expired **Solution**: Run `:RepoHealth` to identify problematic repos

"Cloud sync very slow"

Cause: Large files or poor internet connection **Solution**: Adjust rclone transfer settings or exclude large binary files

Advanced Customization

Excluding Files from Backup

Add patterns to exclude unnecessary files:

```
--exclude "*.tmp"
--exclude "*.DS_Store"
--exclude ".git/objects/**"
--exclude "**/node_modules"
--exclude "**/__pycache__"
```

Adjusting Retention Periods

Modify cleanup commands based on your needs: - More frequent snapshots: Change from 24 hours to 48 hours - Longer archive retention: Increase from 6 months to 1 year - More aggressive cleanup: Reduce thresholds when space is limited

Custom Recovery Scripts

Create project-specific recovery scripts for common scenarios:

```
# Quick recovery of specific project
recover_project() {
    local project_name="$1"
    local source_snapshot="$2"
    rsync -av "$source_snapshot/$project_name/" "~/recovered_$project_name/"
}
```

Security Considerations

Local Security

- External drive uses encrypted HFS+ or APFS
- Scripts run with user permissions (not root)
- Backup logs don't contain sensitive information

Cloud Security

- Use OAuth authentication for cloud services
- Enable two-factor authentication on cloud accounts
- Regularly review cloud service access permissions

Git Security

- Use SSH keys for GitHub authentication
- Regularly rotate authentication tokens
- Monitor for unauthorized repository access

Performance Optimization

Rsync Optimization

The system uses optimized rsync settings:

- `--partial-dir`: Resumes interrupted transfers
- `--inplace`: Reduces disk I/O for large files
- `--compress-level=1`: Light compression for speed

Cloud Sync Optimization

- Multiple transfer threads for faster uploads
- Retry logic for network interruptions
- Bandwidth limiting during business hours (optional)

Space Optimization

- Aggressive exclusion of temporary files
- Compressed archives for long-term storage
- Intelligent cleanup based on available space

Disaster Recovery Planning

Complete System Loss

1. **Immediate**: Access recent work via cloud storage
2. **Short-term**: Clone all repositories from GitHub
3. **Medium-term**: Restore from most recent cloud backup
4. **Long-term**: Restore from monthly archives if needed

External Drive Failure

1. **Immediate**: Continue working (Git + cloud provide protection)
2. **Replace drive**: Purchase new external drive
3. **Restore**: Populate new drive from cloud + Git repositories
4. **Resume**: Automated backups continue with new drive

Cloud Service Issues

1. **Primary cloud down**: Secondary cloud (Dropbox) provides backup
2. **Account access lost**: Git repositories provide version history
3. **Service discontinued**: Export data and configure new provider

Why This System Works for Research

Academic Workflow Compatibility

- **Handles multiple simultaneous projects** - Each with own Git repository
- **Accommodates irregular work patterns** - Automated backups don't depend on routine
- **Supports collaboration** - Individual repos can have different collaborators
- **Preserves history** - Important for reproducible research and publications

Cost-Effectiveness

- **Single external drive** - Minimal hardware investment
- **Free cloud storage tiers** - Often sufficient for code and documents
- **Open source tools** - No licensing costs
- **Scales with needs** - Add more cloud storage or drives as needed

Reliability

- **Multiple independent systems** - Failure of one doesn't compromise others
- **Tested recovery procedures** - Know your recovery works before you need it
- **Automated operation** - Reduces human error

- **Comprehensive monitoring** - Early warning of issues

The system provides enterprise-level data protection using consumer-grade tools, specifically designed for the unique challenges of academic research data management.