Building a Rock-Solid Backup System for Large Research Projects

Research Computing Guide

2025-05-22

Table of contents

1 Building a Rock-Solid Backup System for Large Research Projects

As researchers, our data is our lifeline. When you're managing a 20GB research directory with 300+ subdirectories (like my `~/prj` folder), a simple "drag to cloud" backup just won't cut it. This post walks through building a comprehensive, automated backup system that follows the 3-2-1 rule and integrates seamlessly with a vim-based workflow.

1.1 The Challenge

My research setup: - **Directory**: `~/prj` (20GB, ~300 subdirectories) - **Hardware**: MacBook with two 1TB USB drives ("grey" and "blue") - **Cloud accounts**: iCloud, Google Drive, Dropbox - **Version control**: 300 individual GitHub repositories (one per project) - **Editor**: vim

The goal: hourly automated backups across multiple storage types with easy recovery options.

1.2 Automation vs. Manual Tasks: Understanding Your Daily Workflow

Before diving into the technical setup, it's crucial to understand what this system automates versus what requires your attention. This determines how the backup system integrates into your daily research workflow.

1.2.1 Fully Automated (Set-and-Forget)

These operations run without any intervention once configured:

Hourly: - **Time Machine**: Complete system backups to grey drive - **Local snapshots**: Incremental project backups to grey drive
- **USB mirroring**: Full project sync to blue drive - **Cloud sync**: Project files to Google Drive via rclone

Daily: - **Multi-repo Git sync**: Commits and pushes changes across all 300 repositories (6 PM)

Weekly: - **Compressed archives**: Long-term storage to blue drive (Sunday 2 AM) - **iCloud backup**: Secondary cloud sync (Sunday 8 PM)

1.2.2 Semi-Automated (Triggered by Your Actions)

These happen automatically when you work, but are triggered by your vim usage:

On file save in vim: - **Smart repository sync**: Auto-commits and pushes the current project's repo (throttled to once per 5 minutes per repo) - **Incremental backup**: Triggers local backup if it's been more than 5 minutes since last one

This means: As you work and save files in vim, your individual projects stay backed up to GitHub automatically, without interrupting your flow.

1.2.3 Manual Tasks (Your Regular Routine)

These require your attention and decision-making:

Daily (recommended): - **System health check**: Run `:GitDashboard` in vim or `~/Scripts/git_dashboard.sh` to see which projects need attention - **Review uncommitted changes**: Check if any projects have experimental work that shouldn't be auto-committed

Weekly: - **Repository health check**: Run `:RepoHealth` to identify any problematic repositories - **Drive space monitoring**: Ensure USB drives aren't getting full - **Backup verification**: Spot-check that recent work appears in backups

As needed: - **Recovery operations**: When you need to restore files or projects - **Conflict resolution**: When Git pushes fail due to conflicts - **New project setup**: Adding remotes for new repositories

1.2.4 How Vim Commands Fit Your Daily Routine

The vim integration is designed around natural research workflows:

During active work sessions:

```
" While editing a file, check if this project needs attention
:GitStatus          " Quick check of current project
<leader>gs          " Sync current project immediately (if needed)
```

Daily workflow check (morning routine):

```
" Open vim in your project root and run:
:GitDashboard       " See overview of all 300 projects
:BackupStatus       " Check overall backup system health
```

Weekly maintenance (Friday afternoon):

```
:RepoHealth         " Identify any problematic repositories
:GitSyncAll         " Force sync any repos that might be behind
```

When something seems wrong:

```
:BackupNow          " Force immediate backup
:GitDashboard       " Check what needs attention
```

1.2.5 The "Hands-Off" Philosophy

The system is designed so you can focus on research, not backup management:

1. **Work naturally**: Edit files in vim, save frequently - backups happen automatically
2. **Check periodically**: Quick status checks (30 seconds) show you everything is working
3. **Intervene rarely**: Only when the system alerts you to issues or conflicts

Example daily routine:

```
# Morning: Quick health check (30 seconds)
vim ~/prj/current_project/notes.md
:GitDashboard
# See: "297 clean repos, 3 with changes" - no action needed
```

```
# During work: Just work and save normally
# - Files auto-backup to local drives hourly
# - Git repos auto-sync when you save (throttled)
# - No interruption to your workflow

# End of day: Optional final check
:BackupStatus  # "All systems green" - go home!
```

This approach minimizes cognitive overhead while maximizing protection. You spend less than 5 minutes per week actively managing backups, yet have enterprise-level data protection.

1.3 System Architecture: The 3-2-1 Strategy

The 3-2-1 backup rule forms our foundation: - **3 copies** of data (original + 2 backups) - **2 different storage types** (local + cloud) - **1 offsite backup** (cloud storage)

1.3.1 Storage Allocation Strategy

Grey Drive (1TB): Dual-purpose drive - 800GB: Time Machine (complete system backup) - 200GB: Project snapshots (incremental, hourly)

Blue Drive (1TB): Dedicated project storage - 600GB: Current project mirror - 400GB: Compressed weekly/monthly archives

1.4 Part 1: USB Drive Setup

1.4.1 Partitioning the Drives

First, let's partition our drives for optimal organization:

```
# Grey drive: Time Machine + Project snapshots
diskutil partitionDisk /dev/diskX JHFS+ "TimeMachine" 800G JHFS+ "PrjSnapshots" 200G

# Blue drive: Current mirror + Archives
diskutil partitionDisk /dev/diskY JHFS+ "PrjMirror" 600G JHFS+ "PrjArchive" 400G
```

> **Important**
>
> Replace `/dev/diskX` and `/dev/diskY` with your actual device identifiers. Use `diskutil list` to find them.

1.4.2 Configure Time Machine on Grey Drive

The grey drive's larger partition handles complete system backups:

1. **Set up Time Machine**:
    - Connect the grey drive
    - When prompted, click "Use as Backup Disk" OR:
    - Go to **System Preferences > Time Machine**
    - Click "Select Backup Disk"
    - Choose the "TimeMachine" partition on grey drive
    - Click "Use Disk"
2. **Optimize Time Machine Settings**:

```
# In Time Machine preferences:
#   Back up automatically
#   Back up while on battery power (optional)
```

    - Click "Options" to exclude unnecessary folders:
        - Downloads folder
        - Trash
        - Large temporary directories

> **Note**
>
> Time Machine will now automatically backup your entire system hourly when the grey drive is connected, providing complete system recovery capabilities alongside our project-specific backups.

1.4.3 Setting Up Directory Structure

```
# Create organized directory structure on blue drive
mkdir -p /Volumes/PrjMirror/current
mkdir -p /Volumes/PrjArchive/weekly
mkdir -p /Volumes/PrjArchive/monthly
```

1.5 Part 2: The Master Backup Script

This script handles the heavy lifting of backing up a large directory structure efficiently:

```
#!/bin/bash
# ~/Scripts/prj_backup.sh

LOG_FILE="$HOME/Scripts/backup.log"
PRJ_DIR="$HOME/prj"
GREY_SNAPSHOTS="/Volumes/PrjSnapshots"
BLUE_MIRROR="/Volumes/PrjMirror/current"
TIMESTAMP=$(date "+%Y-%m-%d_%H-%M-%S")

echo "[$TIMESTAMP] Starting enhanced backup process" >> "$LOG_FILE"

# Function to check disk space
check_disk_space() {
    local target_dir="$1"
    local required_gb="25"  # 20GB + 5GB buffer

    if [ -d "$(dirname "$target_dir")" ]; then
        local available_gb=$(df -g "$(dirname "$target_dir")" | tail -1 | awk '{print $4}')
        if [ "$available_gb" -lt "$required_gb" ]; then
            echo "[$TIMESTAMP] WARNING: Low disk space" >> "$LOG_FILE"
            return 1
        fi
    fi
    return 0
}

# Create incremental snapshot on grey drive
create_grey_snapshot() {
    if [ ! -d "/Volumes/PrjSnapshots" ]; then
        echo "[$TIMESTAMP] ERROR: Grey drive not mounted" >> "$LOG_FILE"
        return 1
    fi

    local snapshot_dir="/Volumes/PrjSnapshots/snapshot_$TIMESTAMP"
    mkdir -p "$snapshot_dir"

    # Use rsync with hard links for space efficiency
    local latest_snapshot=$(ls -1t /Volumes/PrjSnapshots/snapshot_* 2>/dev/null | head -1)
```

```
    if [ -n "$latest_snapshot" ] && [ -d "$latest_snapshot" ]; then
        # Incremental backup using hard links
        rsync -av --delete --link-dest="$latest_snapshot" "$PRJ_DIR/" "$snapshot_dir/"
    else
        # First backup
        rsync -av --delete "$PRJ_DIR/" "$snapshot_dir/"
    fi

    # Clean up old snapshots (keep last 48 hours)
    find /Volumes/PrjSnapshots -name "snapshot_*" -type d -mtime +2 -exec rm -rf {} \;
}

# Update blue drive mirror
update_blue_mirror() {
    if [ ! -d "/Volumes/PrjMirror" ]; then
        echo "[$TIMESTAMP] ERROR: Blue drive not mounted" >> "$LOG_FILE"
        return 1
    fi

    check_disk_space "$BLUE_MIRROR" || return 1

    # Full mirror sync with progress
    rsync -av --delete --progress --stats "$PRJ_DIR/" "$BLUE_MIRROR/"
    echo "$TIMESTAMP" > "$BLUE_MIRROR/.last_backup"
}

# Cloud backup with optimizations
update_cloud_backup() {
    if ! command -v rclone >/dev/null 2>&1; then
        echo "[$TIMESTAMP] WARNING: rclone not available" >> "$LOG_FILE"
        return 1
    fi

    rclone sync "$PRJ_DIR/" googledrive:prj/ \
        --exclude "*.tmp" \
        --exclude "*.DS_Store" \
        --exclude ".git/objects/**" \
        --transfers 4 \
        --checkers 8 \
        --retries 3 \
        --log-file="$LOG_FILE"
}
```

```
# Execute all backups
create_grey_snapshot
update_blue_mirror
update_cloud_backup

echo "[$TIMESTAMP] Backup process completed" >> "$LOG_FILE"
```

1.6 Part 3: Automation with launchd

macOS uses launchd for scheduling. Create a launch agent for hourly backups:

```
<!-- ~/Library/LaunchAgents/com.user.prj.backup.plist -->
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
    "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Label</key>
    <string>com.user.prj.backup</string>
    <key>ProgramArguments</key>
    <array>
        <string>/bin/bash</string>
        <string>/Users/YOURUSERNAME/Scripts/prj_backup.sh</string>
    </array>
    <key>StartInterval</key>
    <integer>3600</integer>
    <key>RunAtLoad</key>
    <true/>
    <key>LowPriorityIO</key>
    <true/>
    <key>ProcessType</key>
    <string>Background</string>
</dict>
</plist>
```

Load and start the service:

```
launchctl load ~/Library/LaunchAgents/com.user.prj.backup.plist
launchctl start com.user.prj.backup
```

1.7 Part 4: Multi-Repository Git Management

Having 300 individual GitHub repositories is actually ideal for research projects! Each project maintains its own history, issues, and collaborators. However, this requires a different management strategy.

1.7.1 Repository Discovery and Management

First, let's create a script to discover and manage all repositories:

```
#!/bin/bash
# ~/Scripts/discover_repos.sh

PRJ_DIR="$HOME/prj"
REPO_LIST="$HOME/Scripts/repo_list.txt"

echo "Discovering Git repositories in $PRJ_DIR..."

# Find all .git directories and extract project paths
find "$PRJ_DIR" -name ".git" -type d | while read git_dir; do
    project_dir=$(dirname "$git_dir")
    project_name=$(basename "$project_dir")

    # Get remote origin URL if it exists
    cd "$project_dir"
    remote_url=$(git remote get-url origin 2>/dev/null || echo "NO_REMOTE")

    echo "$project_name|$project_dir|$remote_url"
done > "$REPO_LIST"

repo_count=$(wc -l < "$REPO_LIST")
echo "Found $repo_count Git repositories"
echo "Repository list saved to $REPO_LIST"
```

1.7.2 Bulk Git Operations

Create a script to perform operations across all repositories:

```
#!/bin/bash
# ~/Scripts/bulk_git_ops.sh
```

```
REPO_LIST="$HOME/Scripts/repo_list.txt"
LOG_FILE="$HOME/Scripts/git_bulk.log"
TIMESTAMP=$(date "+%Y-%m-%d %H:%M:%S")

if [ ! -f "$REPO_LIST" ]; then
    echo "Repository list not found. Run discover_repos.sh first."
    exit 1
fi

operation="$1"
if [ -z "$operation" ]; then
    echo "Usage: $0 [status|pull|push|commit|sync]"
    exit 1
fi

echo "[$TIMESTAMP] Starting bulk $operation operation" >> "$LOG_FILE"

while IFS='|' read -r project_name project_dir remote_url; do
    if [ ! -d "$project_dir" ]; then
        continue
    fi

    cd "$project_dir"
    echo "Processing: $project_name"

    case "$operation" in
        "status")
            echo "=== $project_name ===" >> "$LOG_FILE"
            git status --porcelain >> "$LOG_FILE" 2>&1
            ;;
        "pull")
            if [ "$remote_url" != "NO_REMOTE" ]; then
                git pull origin main >> "$LOG_FILE" 2>&1 || \
                git pull origin master >> "$LOG_FILE" 2>&1
            fi
            ;;
        "push")
            if [ "$remote_url" != "NO_REMOTE" ]; then
                git push origin main >> "$LOG_FILE" 2>&1 || \
                git push origin master >> "$LOG_FILE" 2>&1
            fi
            ;;
```

```
        "commit")
            if [ -n "$(git status --porcelain)" ]; then
                git add .
                git commit -m "Automated backup: $TIMESTAMP" >> "$LOG_FILE" 2>&1
                echo "[$TIMESTAMP] $project_name: committed changes" >> "$LOG_FILE"
            fi
            ;;
        "sync")
            # Full sync: commit + push
            if [ -n "$(git status --porcelain)" ]; then
                git add .
                git commit -m "Automated sync: $TIMESTAMP" >> "$LOG_FILE" 2>&1
            fi
            if [ "$remote_url" != "NO_REMOTE" ]; then
                git push origin main >> "$LOG_FILE" 2>&1 || \
                git push origin master >> "$LOG_FILE" 2>&1
            fi
            ;;
    esac
done < "$REPO_LIST"

echo "[$TIMESTAMP] Bulk $operation completed" >> "$LOG_FILE"
```

1.7.3 Multi-Repository Status Dashboard

```
#!/bin/bash
# ~/Scripts/git_dashboard.sh

REPO_LIST="$HOME/Scripts/repo_list.txt"

if [ ! -f "$REPO_LIST" ]; then
    echo "Run discover_repos.sh first to generate repository list"
    exit 1
fi

echo "=== MULTI-REPOSITORY STATUS DASHBOARD ==="
echo "Date: $(date)"
echo ""

uncommitted_count=0
```

```
unpushed_count=0
no_remote_count=0
total_repos=0

echo "Repository Status Summary:"
echo "========================="

while IFS='|' read -r project_name project_dir remote_url; do
    if [ ! -d "$project_dir" ]; then
        continue
    fi

    cd "$project_dir"
    total_repos=$((total_repos + 1))

    # Check for uncommitted changes
    if [ -n "$(git status --porcelain 2>/dev/null)" ]; then
        uncommitted_count=$((uncommitted_count + 1))
        echo "  $project_name: Uncommitted changes"
    fi

    # Check for unpushed commits (if remote exists)
    if [ "$remote_url" != "NO_REMOTE" ]; then
        # Check if ahead of remote
        git fetch origin >/dev/null 2>&1
        ahead=$(git rev-list --count HEAD ^origin/main 2>/dev/null || \
                git rev-list --count HEAD ^origin/master 2>/dev/null || echo "0")

        if [ "$ahead" -gt 0 ]; then
            unpushed_count=$((unpushed_count + 1))
            echo "  $project_name: $ahead commits ahead of remote"
        fi
    else
        no_remote_count=$((no_remote_count + 1))
        echo "  $project_name: No remote configured"
    fi
done < "$REPO_LIST"

echo ""
echo "SUMMARY:"
echo "Total repositories: $total_repos"
echo "With uncommitted changes: $uncommitted_count"
```

```
echo "With unpushed commits: $unpushed_count"
echo "Without remote: $no_remote_count"
echo "Clean repositories: $((total_repos - uncommitted_count - unpushed_count))"
```

1.7.4 Enhanced Backup Script for Multi-Repo Structure

Update the main backup script to work with individual repositories:

```
#!/bin/bash
# ~/Scripts/prj_backup.sh (updated for multiple repos)

LOG_FILE="$HOME/Scripts/backup.log"
PRJ_DIR="$HOME/prj"
GREY_SNAPSHOTS="/Volumes/PrjSnapshots"
BLUE_MIRROR="/Volumes/PrjMirror/current"
TIMESTAMP=$(date "+%Y-%m-%d_%H-%M-%S")

echo "[$TIMESTAMP] Starting multi-repository backup process" >> "$LOG_FILE"

# Update repository list
~/Scripts/discover_repos.sh >/dev/null 2>&1

# Git operations for all repositories
bulk_git_sync() {
    echo "[$TIMESTAMP] Starting bulk Git sync for all repositories" >> "$LOG_FILE"

    # Commit changes in all repos
    ~/Scripts/bulk_git_ops.sh commit >/dev/null 2>&1

    # Push changes (with error handling)
    if ~/Scripts/bulk_git_ops.sh push >/dev/null 2>&1; then
        echo "[$TIMESTAMP] Bulk Git sync completed successfully" >> "$LOG_FILE"
        return 0
    else
        echo "[$TIMESTAMP] Some Git pushes failed - check git_bulk.log" >> "$LOG_FILE"
        return 1
    fi
}

# [Include previous functions: check_disk_space, create_grey_snapshot, update_blue_mirror, et
```

```
# Execute all backups with Git sync
bulk_git_sync
create_grey_snapshot
update_blue_mirror
update_cloud_backup

echo "[$TIMESTAMP] Multi-repository backup process completed" >> "$LOG_FILE"
```

1.7.5 Repository Health Check

```
#!/bin/bash
# ~/Scripts/repo_health_check.sh

REPO_LIST="$HOME/Scripts/repo_list.txt"

echo "=== REPOSITORY HEALTH CHECK ==="

problem_repos=0
healthy_repos=0

while IFS='|' read -r project_name project_dir remote_url; do
    if [ ! -d "$project_dir" ]; then
        continue
    fi

    cd "$project_dir"

    # Check if .git directory is healthy
    if ! git status >/dev/null 2>&1; then
        echo "  $project_name: Git repository corrupted"
        problem_repos=$((problem_repos + 1))
        continue
    fi

    # Check remote connectivity
    if [ "$remote_url" != "NO_REMOTE" ]; then
        if ! git ls-remote origin >/dev/null 2>&1; then
            echo "   $project_name: Remote connectivity issues"
            problem_repos=$((problem_repos + 1))
            continue
```

```
        fi
    fi

    # Check for very old commits (possible stale repos)
    last_commit_date=$(git log -1 --format="%ct" 2>/dev/null)
    if [ -n "$last_commit_date" ]; then
        days_old=$(( ($(date +%s) - last_commit_date) / 86400 ))
        if [ "$days_old" -gt 365 ]; then
            echo "  $project_name: Last commit $days_old days ago"
        fi
    fi

    healthy_repos=$((healthy_repos + 1))

done < "$REPO_LIST"

echo ""
echo "Health Summary:"
echo "Healthy repositories: $healthy_repos"
echo "Problem repositories: $problem_repos"
```

1.8 Part 5: Vim Integration

As a vim user, having backup commands at your fingertips is essential:

```
" ~/.vimrc additions for multi-repository management
command! BackupStatus :!~/Scripts/backup_status.sh
command! BackupNow :!~/Scripts/prj_backup.sh
command! GitDashboard :!~/Scripts/git_dashboard.sh
command! RepoHealth :!~/Scripts/repo_health_check.sh
command! GitSyncAll :!~/Scripts/bulk_git_ops.sh sync

" Project-specific Git operations (for current directory)
command! GitStatus :!git status
command! GitCommit :!git add . && git commit -m "Manual commit from vim"
command! GitPush :!git push origin main || git push origin master
command! GitSync :!git add . && git commit -m "Manual sync from vim" && (git push origin mai

" Quick mappings
nnoremap <leader>bs :BackupStatus<CR>
nnoremap <leader>bn :BackupNow<CR>
```

```
nnoremap <leader>gd :GitDashboard<CR>
nnoremap <leader>gh :RepoHealth<CR>
nnoremap <leader>ga :GitSyncAll<CR>
nnoremap <leader>gs :GitSync<CR>

" Project navigation
nnoremap <leader>pf :find ~/prj/**/*
nnoremap <leader>pg :grep -r "" ~/prj/<Left><Left><Left><Left><Left><Left><Left><Left><Left><

" Smart backup - only sync current project repository
function! SmartGitBackup()
    let current_dir = expand('%:p:h')
    if stridx(current_dir, expand('~/prj')) == 0
        " Find the git root for current file
        let git_root = systemlist('cd ' . shellescape(current_dir) . ' && git rev-parse --sho
        if !empty(git_root) && !v:shell_error
            execute '!cd ' . shellescape(git_root) . ' && git add . && git commit -m "Auto-sa
        endif
    endif
endfunction

" Throttled auto-backup per repository
let g:repo_backup_times = {}
function! ConditionalRepoBackup()
    let current_dir = expand('%:p:h')
    if stridx(current_dir, expand('~/prj')) == 0
        let git_root = systemlist('cd ' . shellescape(current_dir) . ' && git rev-parse --sho
        if !empty(git_root) && !v:shell_error
            let current_time = localtime()
            if !has_key(g:repo_backup_times, git_root) || (current_time - g:repo_backup_times
                let g:repo_backup_times[git_root] = current_time
                call SmartGitBackup()
            endif
        endif
    endif
endfunction

autocmd BufWritePost ~/prj/* call ConditionalRepoBackup()
```

1.9 Part 6: Monitoring and Health Checks

1.9.1 Comprehensive Status Script

```
#!/bin/bash
# ~/Scripts/backup_status.sh

echo "=== PROJECT BACKUP SYSTEM STATUS ==="
echo "Date: $(date)"
echo ""

# Project overview
if [ -d "$HOME/prj" ]; then
    echo "PROJECT OVERVIEW:"
    echo "Size: $(du -sh "$HOME/prj" | cut -f1)"
    echo "Subdirectories: $(find "$HOME/prj" -type d | wc -l | tr -d ' ')"
    echo "Files: $(find "$HOME/prj" -type f | wc -l | tr -d ' ')"
fi

# USB drive status
echo ""
echo "USB DRIVES:"
df -h | grep -E "(PrjSnapshots|PrjMirror|PrjArchive|TimeMachine)" | \
    while read line; do
        echo "  $line"
    done

# Time Machine status
echo ""
echo "TIME MACHINE STATUS:"
tmutil status | grep -E "(Running|Backup|Progress|NextBackup)"
if [ $? -ne 0 ]; then
    echo "  Time Machine: $(tmutil status | head -2 | tail -1)"
fi

# Recent snapshots
echo ""
echo "RECENT SNAPSHOTS:"
if [ -d "/Volumes/PrjSnapshots" ]; then
    ls -lt /Volumes/PrjSnapshots/snapshot_* 2>/dev/null | head -3 | \
        while read line; do
```

```
            dir_name=$(echo "$line" | awk '{print $9}')
            size=$(du -sh "$dir_name" 2>/dev/null | cut -f1)
            echo "  $(basename "$dir_name"): $size"
        done
fi

# Git repository status
echo ""
echo "MULTI-REPOSITORY GIT STATUS:"
if [ -f "$HOME/Scripts/repo_list.txt" ]; then
    total_repos=$(wc -l < "$HOME/Scripts/repo_list.txt")
    echo "Total repositories: $total_repos"

    # Count repositories with uncommitted changes
    uncommitted=0
    while IFS='|' read -r project_name project_dir remote_url; do
        if [ -d "$project_dir" ]; then
            cd "$project_dir"
            if [ -n "$(git status --porcelain 2>/dev/null)" ]; then
                uncommitted=$((uncommitted + 1))
            fi
        fi
    done < "$HOME/Scripts/repo_list.txt"

    echo "Repositories with uncommitted changes: $uncommitted"
    echo "Clean repositories: $((total_repos - uncommitted))"
    echo "Run 'git_dashboard.sh' for detailed status"
else
    echo "Repository list not found - run discover_repos.sh"
fi
```

1.9.2 Health Check Script

```
#!/bin/bash
# ~/Scripts/backup_health.sh

check_backup_integrity() {
    local backup_path="$1"
    local backup_name="$2"
```

```
    if [ ! -d "$backup_path" ]; then
        echo "  $backup_name: Not found"
        return 1
    fi

    local dir_count=$(find "$backup_path" -type d 2>/dev/null | wc -l | tr -d ' ')
    local file_count=$(find "$backup_path" -type f 2>/dev/null | wc -l | tr -d ' ')
    local size=$(du -sh "$backup_path" 2>/dev/null | cut -f1)

    echo "  $backup_name: $dir_count dirs, $file_count files, $size"

    # Sanity check directory count
    if [ "$dir_count" -lt 250 ]; then
        echo "   $backup_name: Directory count seems low ($dir_count < 250)"
    fi
}

echo "=== BACKUP INTEGRITY CHECK ==="
check_backup_integrity "$HOME/prj" "Original Project"

# Check latest snapshot
latest_snapshot=$(ls -1t /Volumes/PrjSnapshots/snapshot_* 2>/dev/null | head -1)
if [ -n "$latest_snapshot" ]; then
    check_backup_integrity "$latest_snapshot" "Latest Snapshot"
fi

check_backup_integrity "/Volumes/PrjMirror/current" "Blue Drive Mirror"
```

1.10 Part 7: Recovery Procedures

1.10.1 Recovery Helper Script

When disaster strikes, you need quick access to recovery options:

```
#!/bin/bash
# ~/Scripts/recovery_helper.sh

echo "=== RECOVERY OPTIONS ==="
echo ""

echo "1. RECENT CHANGES (Grey Drive Snapshots):"
```

```
if [ -d "/Volumes/PrjSnapshots" ]; then
    ls -lt /Volumes/PrjSnapshots/snapshot_* 2>/dev/null | head -5 | \
        while read line; do
            dir_name=$(echo "$line" | awk '{print $9}')
            timestamp=$(basename "$dir_name" | sed 's/snapshot_//')
            size=$(du -sh "$dir_name" 2>/dev/null | cut -f1)
            echo "   $timestamp ($size)"
        done
fi

echo ""
echo "2. COMPLETE MIRROR (Blue Drive):"
if [ -f "/Volumes/PrjMirror/current/.last_backup" ]; then
    last_backup=$(cat /Volumes/PrjMirror/current/.last_backup)
    size=$(du -sh /Volumes/PrjMirror/current 2>/dev/null | cut -f1)
    echo "   Last backup: $last_backup ($size)"
fi

echo ""
echo "=== RECOVERY COMMANDS ==="
echo ""
echo "From latest snapshot:"
echo "  rsync -av /Volumes/PrjSnapshots/snapshot_TIMESTAMP/ ~/prj_recovered/"
echo ""
echo "From blue drive mirror:"
echo "  rsync -av /Volumes/PrjMirror/current/ ~/prj_recovered/"
echo ""
echo "From individual Git repositories:"
echo "  # Clone specific project"
echo "  git clone https://github.com/rgt47/PROJECT_NAME.git ~/recovered/PROJECT_NAME"
echo ""
echo "  # Bulk clone all repositories (if you have a list)"
echo "  while read repo; do"
echo "    git clone https://github.com/rgt47/\$repo.git ~/recovered/\$repo"
echo "  done < repository_names.txt"
echo ""
echo "  # Find and clone repositories from GitHub API"
echo "  curl -s 'https://api.github.com/users/rgt47/repos?per_page=100' | \\"
echo "    jq -r '.[] | select(.name | contains(\"project\")) | .clone_url' | \\"
echo "    while read url; do git clone \$url ~/recovered/; done"
echo ""
echo "From Time Machine (system-wide recovery):"
```

```
echo "  1. Open Time Machine app"
echo "  2. Navigate to ~/prj directory"
echo "  3. Select desired backup date"
echo "  4. Click 'Restore' for files/folders"
echo ""
echo "Command line Time Machine recovery:"
echo "  tmutil listbackups  # List available backups"
echo "  sudo tmutil restore /path/to/backup/prj ~/prj_recovered"
```

1.11 Part 8: Performance Optimization

For a 20GB directory with 300 subdirectories, performance matters:

1.11.1 Optimized rsync Settings

```
# For large directory structures
rsync -av --delete \
    --partial-dir=/tmp/rsync-partial \
    --inplace \
    --compress-level=1 \
    --itemize-changes \
    "$SOURCE/" "$DEST/"
```

1.11.2 rclone Configuration

```
# ~/.config/rclone/rclone.conf
[googledrive]
type = drive
# ... your existing config ...
chunk_size = 64M
upload_cutoff = 64M
```

1.12 Part 9: Weekly Archives

Create compressed archives for long-term storage:

```
#!/bin/bash
# ~/Scripts/weekly_archive.sh

ARCHIVE_DIR="/Volumes/PrjArchive/weekly"
TIMESTAMP=$(date "+%Y-%m-%d")
archive_file="$ARCHIVE_DIR/prj_weekly_$TIMESTAMP.tar.gz"

# Create compressed archive
tar -czf "$archive_file" -C "$HOME" prj/

# Clean up old archives (keep 8 weeks)
find "$ARCHIVE_DIR" -name "prj_weekly_*.tar.gz" -mtime +56 -delete

# Monthly archive (first Sunday of month)
if [ $(date +%d) -le 7 ]; then
    monthly_file="/Volumes/PrjArchive/monthly/prj_monthly_$(date +%Y-%m).tar.gz"
    cp "$archive_file" "$monthly_file"
fi
```

Schedule weekly:

```
(crontab -l 2>/dev/null; echo "0 2 * * 0 $HOME/Scripts/weekly_archive.sh") | crontab -
```

1.13 Quick Reference

1.13.1 Daily Commands

```
# System status
~/Scripts/backup_status.sh

# Multi-repository dashboard
~/Scripts/git_dashboard.sh

# Repository health check
~/Scripts/repo_health_check.sh

# Manual backup
~/Scripts/prj_backup.sh

# Bulk Git operations
```

```
~/Scripts/bulk_git_ops.sh sync    # Commit and push all repos
~/Scripts/bulk_git_ops.sh status  # Check status of all repos
~/Scripts/bulk_git_ops.sh pull    # Pull updates for all repos
```

1.13.2 Vim Commands

Within vim, while editing files in `~/prj`:

```
:BackupStatus    " Check system status
:GitDashboard    " Multi-repository overview
:RepoHealth      " Repository health check
:BackupNow       " Run manual backup
:GitSyncAll      " Sync all repositories
:GitSync         " Sync current repository

<leader>bs       " Quick status
<leader>gd       " Git dashboard
<leader>ga       " Git sync all
<leader>gs       " Git sync current
```

1.13.3 Backup Schedule Summary

Frequency	Action	Storage
Hourly	Time Machine (system), Local snapshots, Cloud sync	Grey drive, Google Drive
Daily	Multi-repo Git sync (commit + push all 300 repos)	GitHub (individual repos)
Weekly	Compressed archives, iCloud	Blue drive, iCloud
Monthly	Long-term archives	Blue drive

1.14 Conclusion

This comprehensive backup system provides multiple layers of protection for large research projects with individual Git repositories per project. The combination of local snapshots,

mirrored drives, individual version control per project, and cloud storage ensures your data is safe from hardware failure, accidental deletion, ransomware, and other disasters.

Key benefits:

- **Project-specific version control**: Each of your 300 projects maintains its own Git history, issues, and collaboration space
- **Automated**: Runs without manual intervention across all repositories
- **Scalable**: Handles large directory structures with hundreds of individual repos efficiently

- **Granular recovery**: Restore individual projects or the entire collection
- **Integrated**: Works seamlessly with vim workflow and individual project development
- **Monitored**: Easy status checking across all repositories with health verification

The multi-repository approach is particularly powerful for research because:

1. **Isolation**: Issues in one project don't affect others
2. **Collaboration**: Each project can have different collaborators and access levels
3. **History**: Detailed commit history per project for publication and reproducibility
4. **Flexibility**: Different projects can use different branching strategies or release cycles
5. **Discoverability**: Individual repos are easier to find and share with colleagues

The system follows industry best practices while being tailored for academic research workflows. With proper setup, you can focus on your research knowing your data is comprehensively protected.

Implementation Tips

1. Start with the basic scripts and test thoroughly before adding automation
2. Monitor backup logs regularly for the first few weeks
3. Test recovery procedures before you need them
4. Adjust schedules based on your actual usage patterns
5. Keep the quick reference handy for daily operations

This system has been tested with macOS and should work with minor modifications on other Unix-like systems. Always test backup and recovery procedures in a safe environment before relying on them for critical data.