SINE NOMINE ASSOCIATES

Git for VMers

Tags, Branches and Commits oh my

Cheyenne Wills
cwills@sinenomine.net

2026 VM Workshop

Disclaimer

SINE NOMINE ASSOCIATES

This talk isn't about running Git on VM.

It's a look at what Git is from the point of view of someone whose first exposure to source code management was on VM many years ago.

Topics

SINE NOMINE ASSOCIATES

- The world according to CNTRL, AUX and update files
- The journey from file cabinets to being scattered across the globe
- Eek! Waiter, there's a bug in my code, I would like to know how it got there
  - Oh.. you already fixed it!
- Branches are cheap
- Will the real clone please stand up
- Keeping up with main
- Now where did you say my files are?

SINE NOMINE ASSOCIATES

VMSES/E

- Software Inventory
  - Which products are installed
  - Status of maintenance and modifications
- Package Manager
  - Product layout
  - Requisite management
- Patch Management
  - Receives and applies patches
- Object Builder
  - Creates the usable objects (modules, TXTLIBs, MACLIBs, etc.)

SINE NOMINE ASSOCIATES

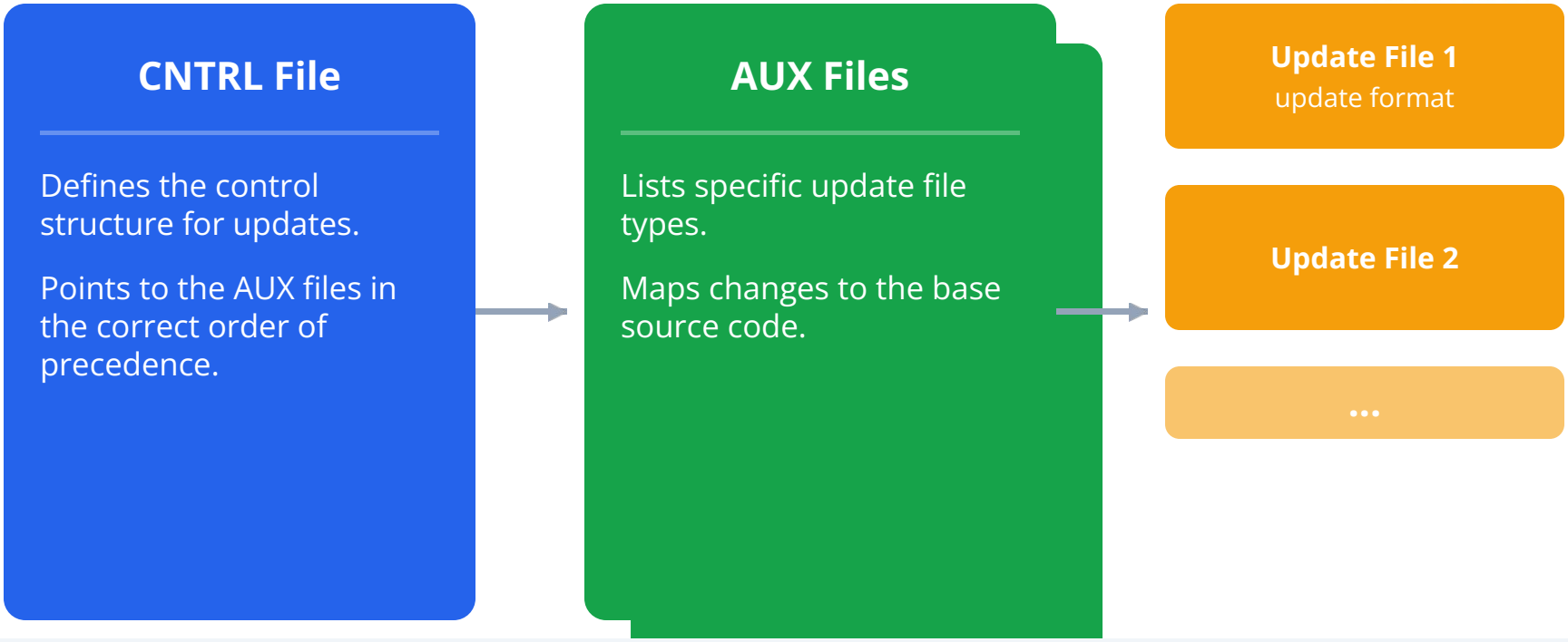
VMSES/E

- Software Inventory
  - Which products are installed
  - Status of maintenance and modifications
- Package Manager
  - Product layout
  - Requisite management
- Patch Management
  - Receives and applies patches

- Object Builder
  - Creates the usable objects (modules, TXTLIBs, MACLIBs, etc.)

VM Maintenance: File Relationships

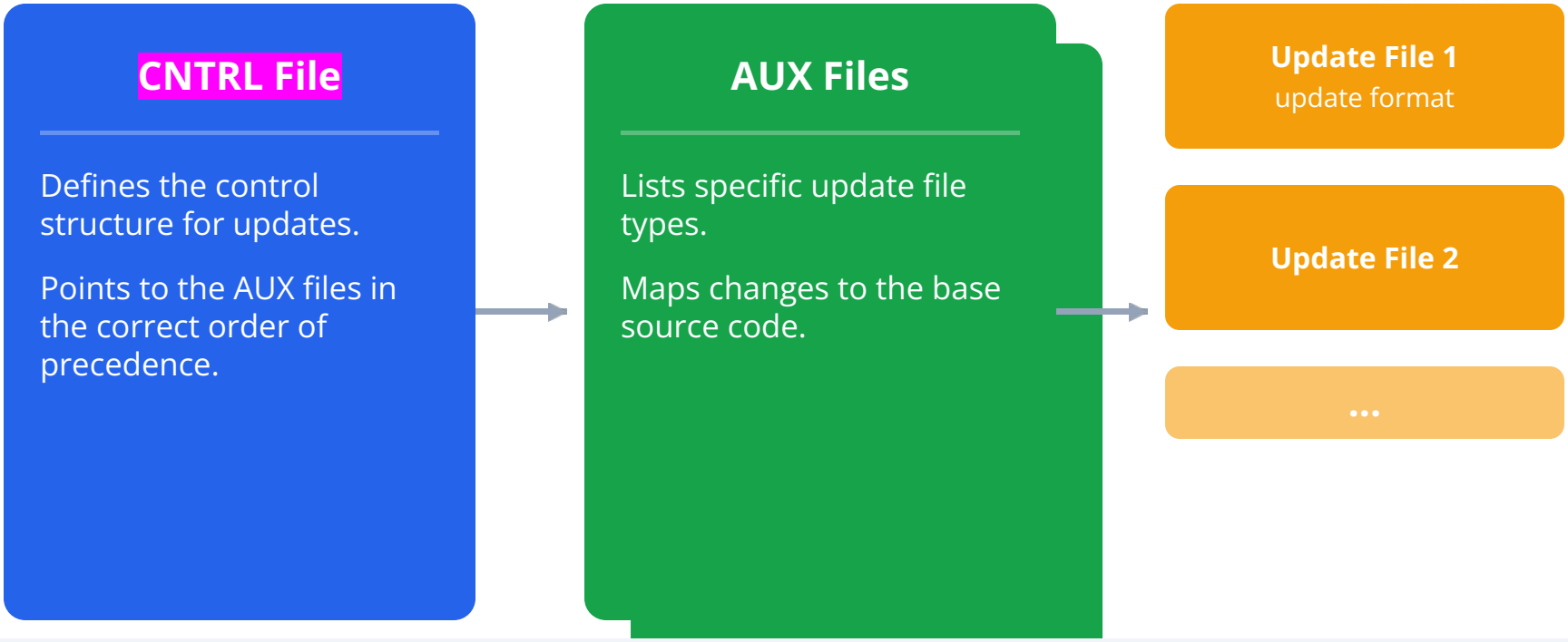
SINE NOMINE ASSOCIATES



Maintenance files work together to manage source modifications

VM Maintenance: File Relationships

SINE NOMINE ASSOCIATES



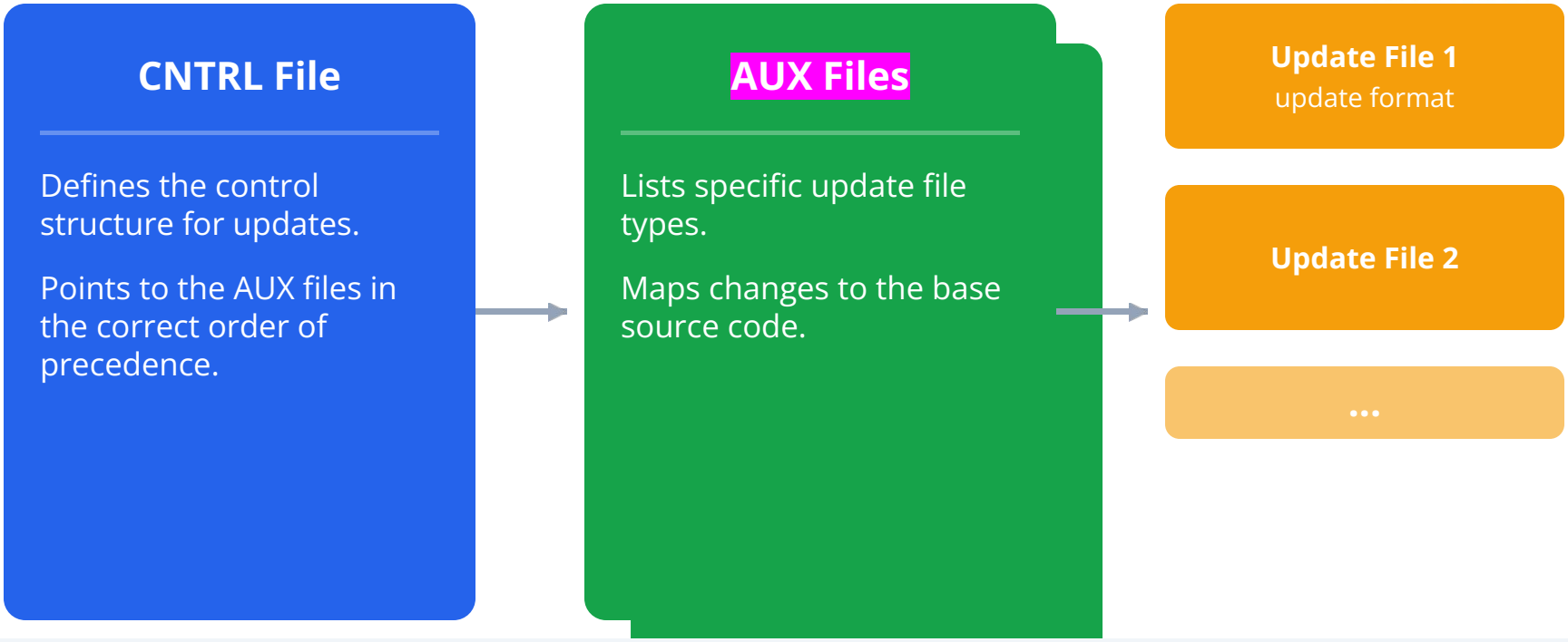
Maintenance files work together to manage source modifications

The CNTRL file establishes the maintenance hierarchy, clearly defining the precedence for AUX files:

- scheduled maintenance
- unscheduled maintenance fixes
- and local modifications.

VM Maintenance: File Relationships

SINE NOMINE ASSOCIATES

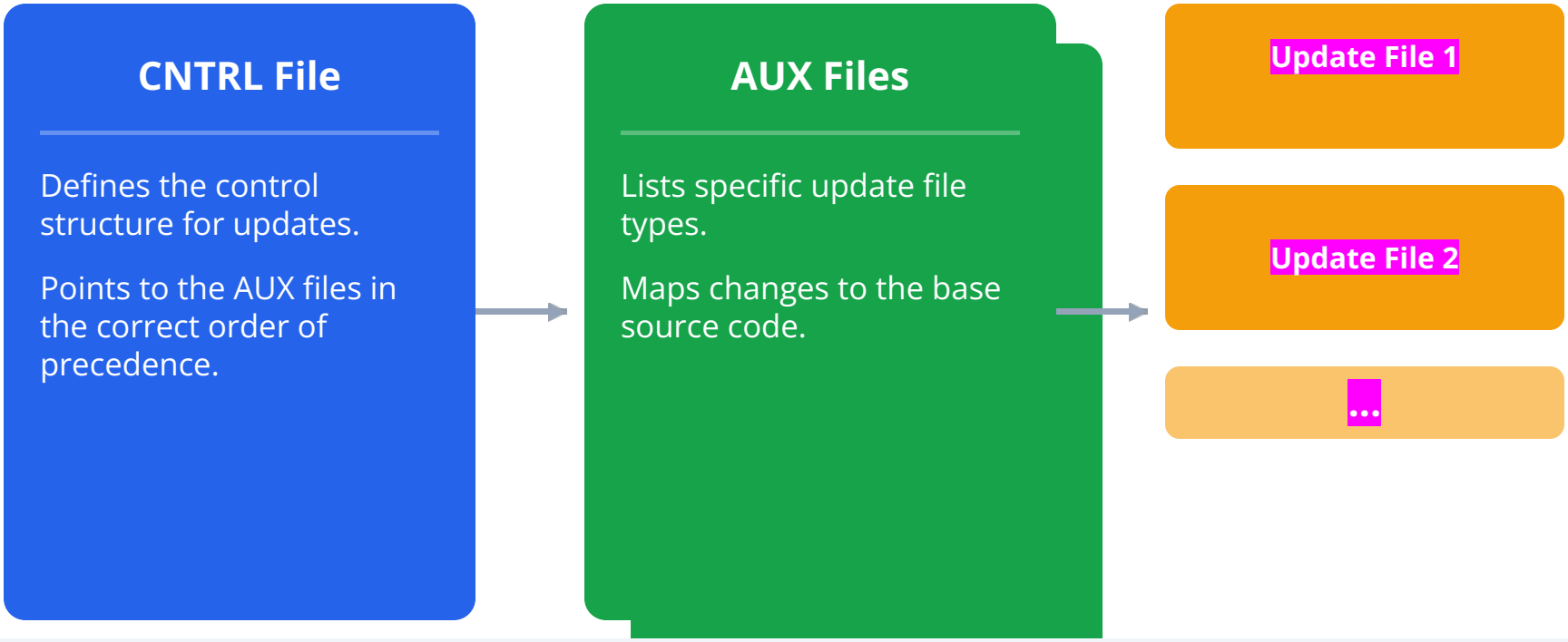


Maintenance files work together to manage source modifications

AUX files share the filename of the source file and can be used to see the change history for that particular file.

VM Maintenance: File Relationships

SINE NOMINE ASSOCIATES



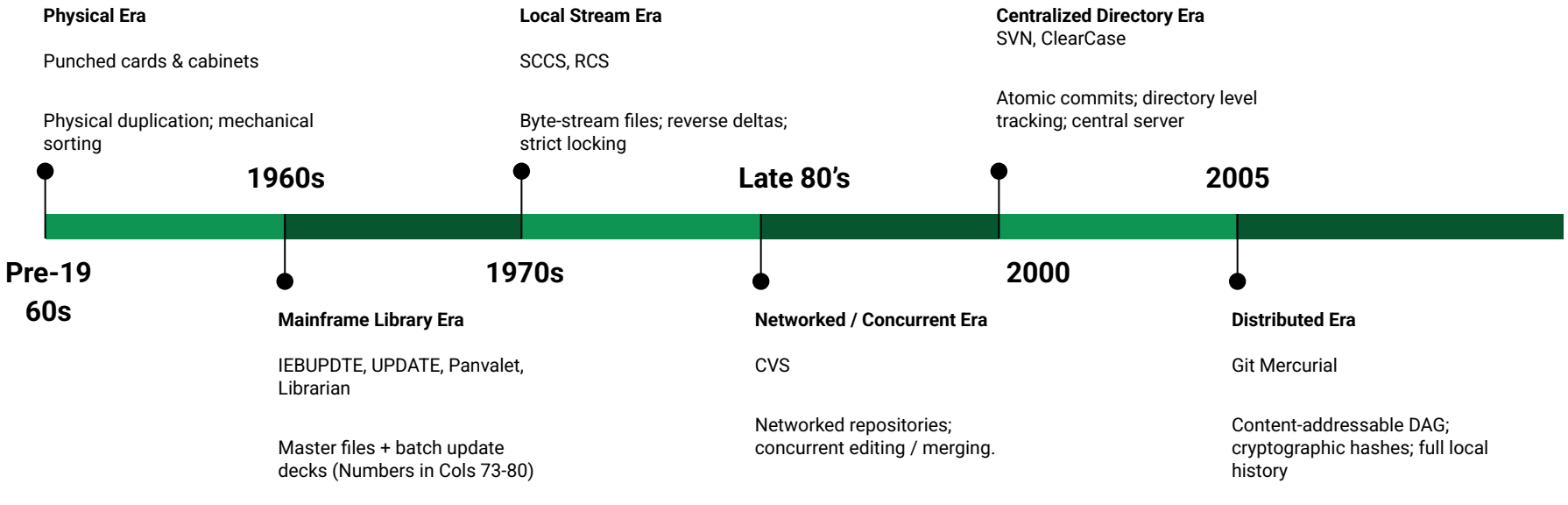
Maintenance files work together to manage source modifications

Update files contain the actual source modifications. They mirror the source filename, and their specific filetype is cataloged inside an associated AUX file, where consistent naming conventions are essential for grouping related changes.

VM Maintenance File Relationships

The journey from file cabinets to being scattered across the globe

SINE NOMINE ASSOCIATES



What is Git?

SINE NOMINE ASSOCIATES

A Version Control Tool: Git is a command-line tool that systematically records changes to a set of files over time, allowing you to recall or revert to specific versions at will.

- **The Local Safety Net:** It allows developers to edit, delete, or experiment with source files, knowing that any previous state of the project can be instantly restored.
- **The Collaboration Engine:** It provides a structured mechanism for multiple developers to safely merge their independent code changes into a single, cohesive project without overwriting each other's work.
- **The Historical Ledger:** It permanently logs exactly who made a change, when they made it, and the stated reason for the modification.

What is Git?

SINE NOMINE ASSOCIATES

Starting from Scratch: `git init`

- **The Scenario:** You have a local directory containing source code or scripts that are not currently under version control, and you want to start tracking them.
- **Step 1: Initialize.** Running `git init` inside your project directory creates a hidden `.git` folder. This is the empty database that will store your project's history.
- **Step 2: Stage.** Running `git add .` tells Git to take a snapshot of all the current files in the directory and place them into the staging area.
- **Step 3: Commit.** Running `git commit -m "Initial project import"` permanently writes that staged snapshot into the database as your very first historical record.

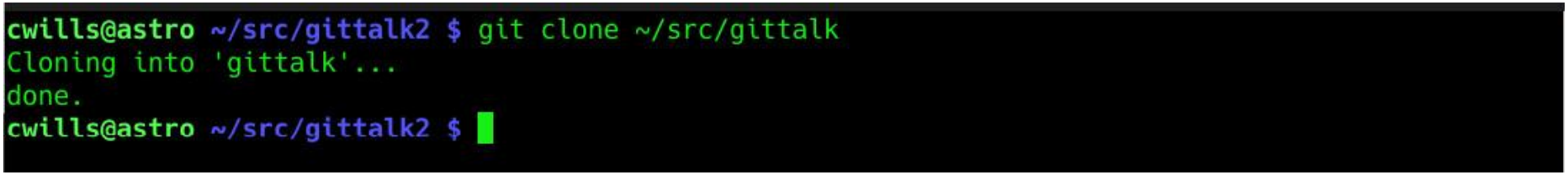
```
cwills@astro ~/src/gittalk $ git init
Initialized empty Git repository in /home/cwills/src/gittalk/.git/
cwills@astro ~/src/gittalk $ echo "Hello World" > readme.txt
cwills@astro ~/src/gittalk $ git add .
cwills@astro ~/src/gittalk $ git commit -m "Initial project import"
[main (root-commit) 1b4035a] Initial project import
 1 file changed, 1 insertion(+)
 create mode 100644 readme.txt
cwills@astro ~/src/gittalk $
```

What is Git?

SINE NOMINE ASSOCIATES

Joining an Existing Project: `git clone`

- **The Scenario:** The project already exists on a remote server (like GitHub, GitLab, or an internal corporate server), and you need to download it to your workstation to start contributing.
- **The Command:** You execute `git clone <repository-URL>` from your terminal.
- **The Execution:** Git creates a new directory, connects to the remote server, and downloads the repository data.
- **The Result:** You do not just receive the latest source files. A clone downloads the **entire historical database**. Your local machine now possesses every commit, every branch, and every file version that has ever existed in the project.



What is Git?

SINE NOMINE ASSOCIATES

Joining an Existing Project: `git clone`

- **The Scenario:** The project already exists on a remote server (like GitHub, GitLab, or an internal corporate server), and you need to download it to your workstation to start contributing.
- **The Command:** You execute `git clone <repository-URL>` from your terminal.
- **The Execution:** Git creates a new directory, connects to the remote server, and downloads the repository data.
- **The Result:** You do not just receive the latest source files. A clone downloads the **entire historical database**. Your local machine now possesses every commit, every branch, and every file version that has ever existed in the project.

What is Git?

SINE NOMINE ASSOCIATES

The Local-First Workflow

- **Complete Independence:** When you clone a repository, you download the entire history of the project. You are not just linking to a remote server; you possess a full, standalone copy of the system.
- **Offline Operation:** Because the history is local, you can review past changes, create new branches, and commit code while entirely disconnected from a network.
- **Zero Latency:** Standard operations—like switching contexts, comparing file versions, or searching logs—execute instantly because they do not require a round-trip to a central server.

SINE NOMINE ASSOCIATES

The Broad Search: `git log`

- **Purpose:** A high-level auditing tool to filter and search the entire repository history.
- `git log -S "string"` **(The Pickaxe):** Searches the raw diffs to find exactly when a specific variable or function was added or removed.
- `git log --grep="keyword"`**:** Filters the output to only show commits containing specific keywords in their commit messages.
- `git log -p`**:** Expands the log to display the actual line-by-line patch applied in each commit.

```
cwills@astro ~/src/myopenafs_repo $ git -P log  --oneline -S "rx_opaque_cmp"
2e8d1540b rx: Introduce rx_opaque_cmp(), _stringify()
cwills@astro ~/src/myopenafs_repo $ git -P log --oneline --grep "folio_batch.h"
d47c438ae Linux: pagevec.h renamed to folio_batch.h
```

Eek! Waiter, there's a bug in my code, I would like to know how it got there

SINE NOMINE ASSOCIATES

```
cwills@astro ~/src/myopenafs_repo $ git -P log -p -1
commit 12b794b77441c0489027da0ae5acaac8ef001e47 (HEAD -> master, origin/master, origin/HEAD)
Author: Michael Meffie <mmeffie@sinenomine.net>
Date:   Mon Jun 8 15:06:26 2026 -0400

    afs: Remove set but unused cnt in afs_GetDownDSlot()

    The `cnt` variable is incremented but never accessed in
    afs_GetDownDSlot(). Remove the unused variable.

    This fixes a build warning on newer compilers, such as GCC 16.1.1.

        .../src/afs/afs_dcache.c: In function 'afs_GetDownDSlot':
        .../src/afs/afs_dcache.c:1254:18: error: variable 'cnt' set but not used [-Werror=unused-but
-set-variable=]
        |     unsigned int cnt;
        |                  ^~~

    This bug has been present since OpenAFS 1.0.

    Change-Id: I228c7a1b00651fdd38ad946386856865ea126753
    Reviewed-on: https://gerrit.openafs.org/16820
    Tested-by: BuildBot <buildbot@rampaginggeek.com>
    Reviewed-by: Cheyenne Wills <cwills@sinenomine.net>
    Reviewed-by: Michael Meffie <mmeffie@sinenomine.net>

diff --git a/src/afs/afs_dcache.c b/src/afs/afs_dcache.c
index 5caa2a31f..fd4ed91b4 100644
--- a/src/afs/afs_dcache.c
+++ b/src/afs/afs_dcache.c
@@ -1251,7 +1251,6 @@ afs_GetDownDSlot(int anumber)
     struct afs_q *tq, *nq;
     struct dcache *tdc;
     int ix;
-    unsigned int cnt;

     AFS_STATCNT(afs_GetDownDSlot);
```

SINE NOMINE ASSOCIATES

The Line-Level Audit: `git blame`

- **Purpose:** A targeted investigation tool that annotates a specific file line-by-line.
- **Identification:** Displays the exact commit hash, author, and timestamp for the last modification of every single line.
- **Precision:** Instantly pinpoints the precise commit that introduced flawed logic.
- **Context:** Provides the necessary commit hash to investigate the broader scope of why a specific change was made.

Eek! Waiter, there's a bug in my code, I would like to know how it got there

```
cwills@astro ~/src/linux/arch/s390/kernel $ git -P blame facility.c
682dbf430d27b (Nina Schoetterl-Glausch 2023-12-19 15:08:51 +0100  1) // SPDX-License-Identifier: GPL
-2.0
682dbf430d27b (Nina Schoetterl-Glausch 2023-12-19 15:08:51 +0100  2) /*
682dbf430d27b (Nina Schoetterl-Glausch 2023-12-19 15:08:51 +0100  3)  * Copyright IBM Corp. 2023
682dbf430d27b (Nina Schoetterl-Glausch 2023-12-19 15:08:51 +0100  4)  */
682dbf430d27b (Nina Schoetterl-Glausch 2023-12-19 15:08:51 +0100  5)
65c9a9f925024 (Heiko Carstens          2025-06-12 13:47:38 +0200  6) #include <linux/export.h>
682dbf430d27b (Nina Schoetterl-Glausch 2023-12-19 15:08:51 +0100  7) #include <asm/facility.h>
682dbf430d27b (Nina Schoetterl-Glausch 2023-12-19 15:08:51 +0100  8)
682dbf430d27b (Nina Schoetterl-Glausch 2023-12-19 15:08:51 +0100  9) unsigned int stfle_size(void)
682dbf430d27b (Nina Schoetterl-Glausch 2023-12-19 15:08:51 +0100 10) {
682dbf430d27b (Nina Schoetterl-Glausch 2023-12-19 15:08:51 +0100 11)     static unsigned int size;
682dbf430d27b (Nina Schoetterl-Glausch 2023-12-19 15:08:51 +0100 12)     unsigned int r;
682dbf430d27b (Nina Schoetterl-Glausch 2023-12-19 15:08:51 +0100 13)     u64 dummy;
682dbf430d27b (Nina Schoetterl-Glausch 2023-12-19 15:08:51 +0100 14)
682dbf430d27b (Nina Schoetterl-Glausch 2023-12-19 15:08:51 +0100 15)     r = READ_ONCE(size);
682dbf430d27b (Nina Schoetterl-Glausch 2023-12-19 15:08:51 +0100 16)     if (!r) {
682dbf430d27b (Nina Schoetterl-Glausch 2023-12-19 15:08:51 +0100 17)             r = __stfle_asm(&dum
my, 1) + 1;
682dbf430d27b (Nina Schoetterl-Glausch 2023-12-19 15:08:51 +0100 18)             WRITE_ONCE(size, r);
682dbf430d27b (Nina Schoetterl-Glausch 2023-12-19 15:08:51 +0100 19)     }
682dbf430d27b (Nina Schoetterl-Glausch 2023-12-19 15:08:51 +0100 20)     return r;
682dbf430d27b (Nina Schoetterl-Glausch 2023-12-19 15:08:51 +0100 21) }
682dbf430d27b (Nina Schoetterl-Glausch 2023-12-19 15:08:51 +0100 22) EXPORT_SYMBOL(stfle_size);
cwills@astro ~/src/linux/arch/s390/kernel $
```

Eek! Waiter, there's a bug in my code, I would like to know how it got there

SINE NOMINE ASSOCIATES

The Algorithmic Hunt: `git bisect`

- **Purpose:** Automates a binary search through commit history to locate a hidden regression.
- **Setup:** Requires the developer to flag the current broken state as "bad" and a historical working state as "good."
- **Execution:** Calculates the exact midpoint commit between the two states and checks it out for testing.
- **Resolution:** Systematically halves the search space based on test results to rapidly isolate the single breaking commit.

```
cwills@astro ~/src/gittalk $ ./demo.py 5 1
5/1 = 5.0
5 > 1
cwills@astro ~/src/gittalk $ ./demo.py 2 5
2/5 = 0.4
2 < 5
cwills@astro ~/src/gittalk $ ./demo.py 2 0
Traceback (most recent call last):
  File "/home/cwills/src/gittalk/./demo.py", line 10, in <module>
    print(f"{n1}/{n2} = {int(n1) / int(n2)}")
                        ~~~~~~~~~^~~~~~~~~
ZeroDivisionError: division by zero
```

Eek! Waiter, there's a bug in my code, I would like to know how it got there

SINE NOMINE ASSOCIATES

```
f255388 Fix display
3fc15b5 Add new function
1b22841 Add demo program
03c8dc4 Initial project import
cwills@astro ~/src/gittalk $ git bisect start 38cfdd9
status: waiting for good commit(s), bad commit known
cwills@astro ~/src/gittalk $ qit -P log --oneline
bash: qit: command not found
cwills@astro ~/src/gittalk $ git -P log --oneline
38cfdd9 (HEAD -> main) Move divide ahead of checks
f255388 Fix display
3fc15b5 Add new function
1b22841 Add demo program
03c8dc4 Initial project import
cwills@astro ~/src/gittalk $ git bisect next
warning: bisecting only with a bad commit
Are you sure [Y/n]? y
Bisecting: 2 revisions left to test after this (roughly 1 step)
[1b22841cd61129402cea56e589f216865fe3d7e9] Add demo program
cwills@astro ~/src/gittalk $ git -P log --oneline
1b22841 (HEAD) Add demo program
03c8dc4 Initial project import
cwills@astro ~/src/gittalk $ ls
demo.py  readme.txt
cwills@astro ~/src/gittalk $ ./demo.py 2 0
2 > 0
cwills@astro ~/src/gittalk $ git bisect good
Bisecting: 0 revisions left to test after this (roughly 1 step)
[f25538825871310919d4b15782f4087c69e25e1a] Fix display
cwills@astro ~/src/gittalk $ ./demo.py 2 0
2 > 0
Traceback (most recent call last):
  File "/home/cwills/src/gittalk/./demo.py", line 14, in <module>
    print(f"{n1}/{n2} = {int(n1) / int(n2)}")
                         ~~~~~~~~~^~~~~~~~~
ZeroDivisionError: division by zero
cwills@astro ~/src/gittalk $ git bisect bad
Bisecting: 0 revisions left to test after this (roughly 0 steps)
[8fc15b565dfb34f45ac2880ade43bd88bc8b703c] Add new function
cwills@astro ~/src/gittalk $ ./demo.py 2 0
2 > 0
2/0 = 2 / 0
```

Eek! Waiter, there's a bug in my code, I would like to know how it got there

```
cwills@astro ~/src/gittalk $ ./demo.py 2 0
2 > 0
Traceback (most recent call last):
  File "/home/cwills/src/gittalk/./demo.py", line 14, in <module>
    print(f"{n1}/{n2} = {int(n1) / int(n2)}")
                        ~~~~~~~~~^~~~~~~~~
ZeroDivisionError: division by zero

cwills@astro ~/src/gittalk $ git bisect bad
Bisecting: 0 revisions left to test after this (roughly 0 steps)
[8fc15b565dfb34f45ac2880ade43bd88bc8b703c] Add new function

cwills@astro ~/src/gittalk $ ./demo.py 2 0
2 > 0
2/0 = 2 / 0
cwills@astro ~/src/gittalk $ git bisect good
f255388258713109l9d4b15782f4087c69e25e1a is the first bad commit
commit f255388258713109l9d4b15782f4087c69e25e1a
Author: Cheyenne Wills <cheyenne.wills@gmail.com>
Date:   Thu Jun 11 13:03:16 2026 -0600

    Fix display

 demo.py | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Eek! Waiter, there's a bug in my code, I would like to know how it got there

SINE NOMINE ASSOCIATES

```
cwills@astro ~/src/gittalk $ git bisect log
# bad: [38cfdd9c2f686767879e9c38efc0012df6e8755d] Move divide ahead of checks
git bisect start '38cfdd9'
# status: waiting for good commit(s), bad commit known
# good: [1b22841cd61129402cea56e589f216865fe3d7e9] Add demo program
git bisect good 1b22841cd61129402cea56e589f216865fe3d7e9
# bad: [f25538825871310919d4b15782f4087c69e25e1a] Fix display
git bisect bad f25538825871310919d4b15782f4087c69e25e1a
# good: [8fc15b565dfb34f45ac2880ade43bd88bc8b703c] Add new function
git bisect good 8fc15b565dfb34f45ac2880ade43bd88bc8b703c
# first bad commit: [f25538825871310919d4b15782f4087c69e25e1a] Fix display
cwills@astro ~/src/gittalk $ git bisect reset
Previous HEAD position was 8fc15b5 Add new function
Switched to branch 'main'
cwills@astro ~/src/gittalk $
```

Oh.. you already fixed it!

SINE NOMINE ASSOCIATES

The Git Patch

- **Purpose:** Exchanging code changes when a shared network or central repository is unavailable.
- **`git format-patch`:** Extracts commits into standard text files containing the code diff and all associated metadata (author, timestamp, commit message).

```
cwills@astro ~/src/gittalk2/gittalk $ git format-patch -1
0001-handle-condtion-when-2nd-parm-is-zero.patch
cwills@astro ~/src/gittalk2/gittalk $ cp 0001-handle-condtion-when-2nd-parm-is-zero.patch /tmp
cwills@astro ~/src/gittalk2/gittalk $
```

Oh.. you already fixed it!

SINE NOMINE ASSOCIATES

The Git Patch

- **`git apply`:** Reads the patch file and modifies your local working directory, leaving the changes uncommitted for review.
- **`git am`:** Applies the patch file and automatically creates the commit, perfectly preserving the original author's identity and history.

```
cwills@astro ~/src/gittalk $ git am /tmp/0001-handle-condtion-when-2nd-parm-is-zero.patch
Applying: handle condtion when 2nd parm is zero
cwills@astro ~/src/gittalk $
```

Oh.. you already fixed it!

SINE NOMINE ASSOCIATES

Remotes and Fetching

- **Purpose:** Accessing another developer's repository or an upstream server directly.
- **`git remote add`:** Configures a local pointer to an external repository URL (e.g., a colleague's fork or a project's main server).
- **`git fetch`:** Downloads the new commits, files, and branch history from the remote repository into your local `.git` object store.
- **Safety:** Fetching is a read-only network operation. It updates your local tracking data but strictly isolates the downloaded changes from your active working directory.

Oh.. you already fixed it!

SINE NOMINE ASSOCIATES

Integrating the Fix: Merging and Pulling

- **Purpose:** Combining the newly acquired remote fix with your active local code.
- **`git merge`:** Joins the downloaded remote branch with your current local branch, calculating the diffs and creating a new merge commit.
- **`git rebase`:** Temporarily detaches your local commits, applies the remote fix, and then replays your local commits on top to maintain a strictly linear history.
- **`git pull`:** A high-level macro command that executes a `git fetch` and immediately triggers a `git merge` in a single step.

Oh.. you already fixed it!

SINE NOMINE ASSOCIATES

```
cwills@astro ~/src/gittalk $ git remote add myotherrepo /home/cwills/src/gittalk2/gittalk
cwills@astro ~/src/gittalk $ git fetch myotherrepo
From /home/cwills/src/gittalk2/gittalk
 * [new branch]      main       -> myotherrepo/main
cwills@astro ~/src/gittalk $ git branch -al
cwills@astro ~/src/gittalk $ git -P branch -al
* main
  remotes/myotherrepo/HEAD -> myotherrepo/main
  remotes/myotherrepo/main
cwills@astro ~/src/gittalk $ git pull myotherrepo main
From /home/cwills/src/gittalk2/gittalk
 * branch            main       -> FETCH_HEAD
Updating 38cfdd9..eeb1a19
Fast-forward
 demo.py | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
cwills@astro ~/src/gittalk $ git -P log --oneline
eeb1a19 (HEAD -> main, myotherrepo/main, myotherrepo/HEAD) handle condtion when 2nd parm is zero
38cfdd9 Move divide ahead of checks
f255388 Fix display
3fc15b5 Add new function
1b22841 Add demo program
03c8dc4 Initial project import
cwills@astro ~/src/gittalk $
```

Branches are cheap

SINE NOMINE ASSOCIATES

The Mechanics of a "Cheap" Branch

- **The Misconception:** Branching does not physically copy source files or duplicate directories.
- **The Reality:** A Git branch is simply a lightweight text file containing the 40-character SHA-1 hash of a specific commit.
- **The Cost:** Creating a branch requires milliseconds of CPU time and consumes virtually zero disk space.
- **The Result:** One can generate dozens of local branches simultaneously without degrading system performance or storage capacity.

Branches are cheap

SINE NOMINE ASSOCIATES

Why We Branch: Isolation and Safety

- **Feature Development:** Encapsulates new code, ensuring incomplete or untested work never destabilizes the primary timeline.
- **Concurrent Hotfixes:** Allows developers to instantly pivot to a stable production state to build a patch, completely isolated from their in-progress feature work.
- **Disposable Experiments:** Provides a sandbox for testing high-risk architectural changes that can be instantly abandoned and deleted without cleanup.
- **Code Review:** Establishes a defined boundary for peer review and automated testing before changes are authorized for integration.

Branches are cheap

SINE NOMINE ASSOCIATES

How We Branch:

- **`git branch <name>`:** Creates the new branch pointer at the current commit but leaves your active working directory unchanged.
- **`git switch <name>`:** Moves the active `HEAD` pointer to the specified branch and automatically updates your local files to match that state.
- **`git switch -c <name>`:** The standard daily shortcut that creates the new branch and immediately switches to it in a single command.
- **The Lifecycle:** Create the temporary branch, commit isolated changes, merge the verified code back into the main timeline, and delete the branch pointer.

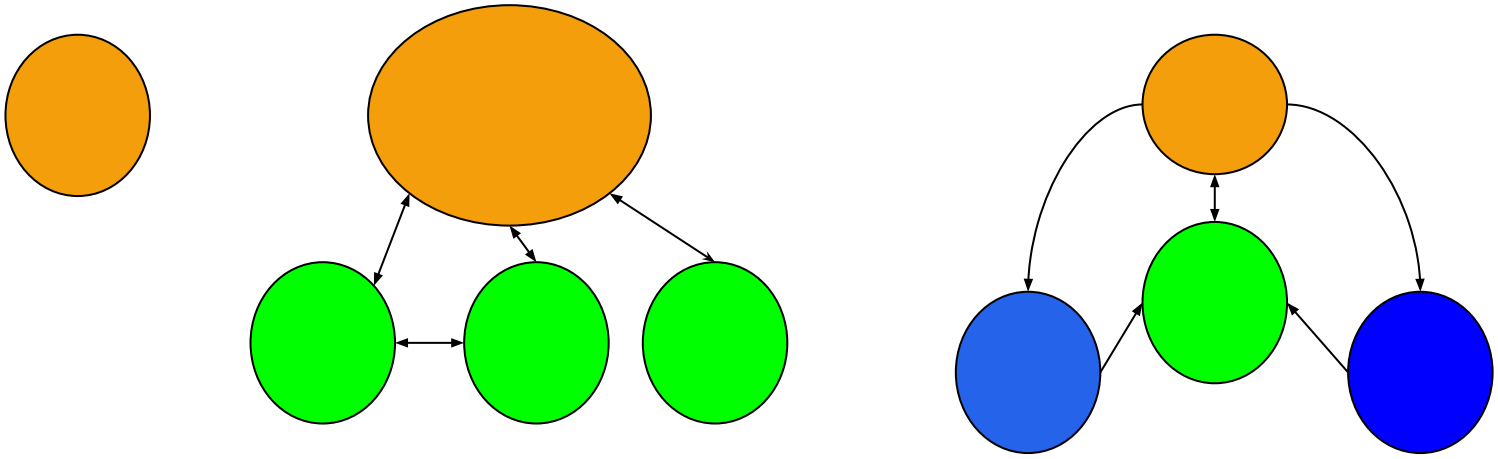
```
cwills@astro ~/src/gittalk $ git -P branch -al
* main
  remotes/myotherrepo/HEAD -> myotherrepo/main
  remotes/myotherrepo/main
cwills@astro ~/src/gittalk $ git switch -c WIP
Switched to a new branch 'WIP'
cwills@astro ~/src/gittalk $ git -P branch -al
* WIP
  main
  remotes/myotherrepo/HEAD -> myotherrepo/main
  remotes/myotherrepo/main
cwills@astro ~/src/gittalk $ git -P log --oneline
eeb1a19 (HEAD -> WIP, myotherrepo/main, myotherrepo/HEAD, main) handle condtion when 2nd parm is ze
o
38cfdd9 Move divide ahead of checks
f255388 Fix display
8fc15b5 Add new function
1b22841 Add demo program
03c8dc4 Initial project import
cwills@astro ~/src/gittalk $
```

Will the real clone please stand up

SINE NOMINE ASSOCIATES

The Source of Truth

- **Solo Developer:** The local clone on the workstation is the absolute authority. Workflows are entirely local, and branch management is for personal organization.
- **Small Teams:** Introduction of a single shared remote repository as the primary synchronization point. Authority relies on basic branch protection and team communication.
- **Enterprise Scale:** The "Golden Master" repository is defined entirely by organizational policy. Direct write access is restricted, and integration is governed by automated testing pipelines and/or strict review mandates.

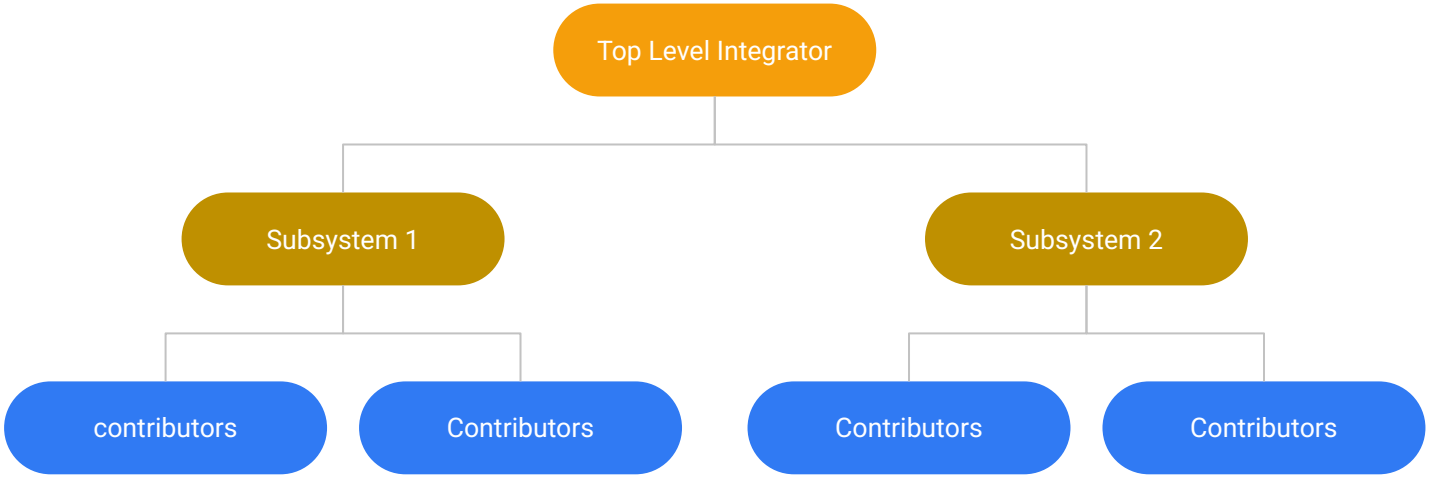


Will the real clone please stand up

SINE NOMINE ASSOCIATES

The Integration Hierarchy (using the Linux development workflow as an example model)

- **The Model:** A distributed chain of trust used to manage massive scale.
- **The Contributor Level:** Developers execute work in isolated, personal forks and push their commits upward via patches or pull requests.
- **The Subsystem Maintainer:** Domain-specific reviewers validate and merge contributor code into dedicated, intermediate subsystem repositories.
- **The Top-Level Integrator:** Final merge authority rests with a core group. They do not review individual lines of code, but rather pull fully integrated and tested subsystem blocks into the definitive mainline repository.

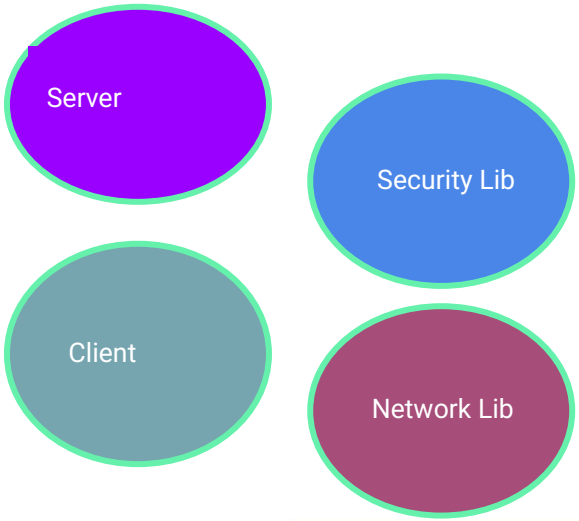


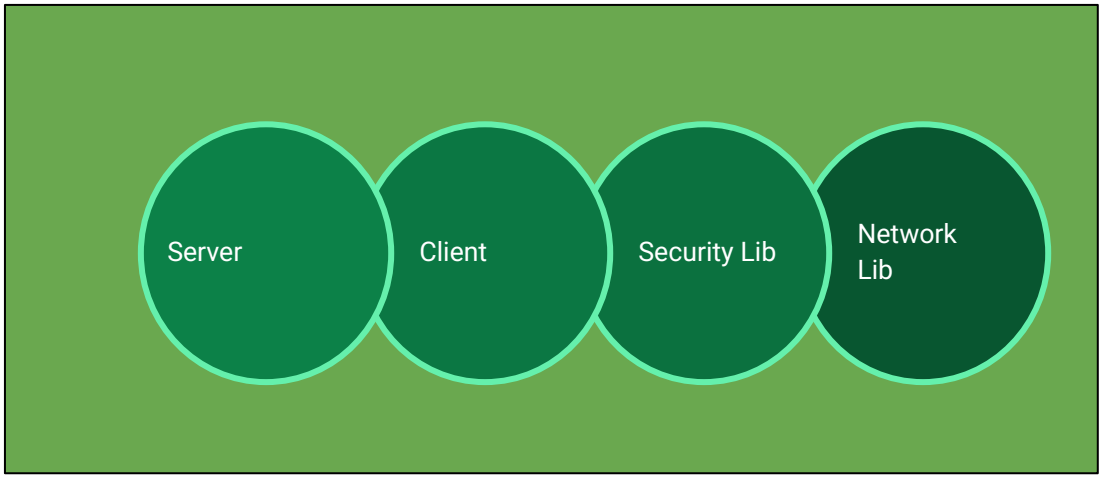
Will the real clone please stand up

SINE NOMINE ASSOCIATES

Monorepos vs. Single-Project Repositories

- **Single-Project Repositories (Polyrepos):** One repository per discrete service or library. Enforces strict modular boundaries and keeps clone sizes minimal, but complicates the management of cross-repository dependencies.
- **The Monorepo:** A single, comprehensive repository containing all projects, services, and libraries for an entire organization.
- **Unified Versioning:** A monorepo guarantees that a single commit can atomically update a core library and all of its dependent projects simultaneously.
- **The Scaling Challenge:** At enterprise scale, monorepos require specialized tooling to manage network checkout sizes, sparse checkouts, and incremental build times.





Keeping up with main

SINE NOMINE ASSOCIATES

Continuous Integration: The Case for Frequency

- **The Cost of Isolation:** Prolonged branch isolation exponentially increases the probability, scale, and complexity of integration conflicts.
- **Incremental Synchronization:** Frequently fetching and integrating the mainline branch into local development streams identifies architectural collisions early.
- **Context Retention:** Resolving conflicts in small, daily increments allows developers to address code overlaps while their immediate logic and context are still fresh.
- **Operational Discipline:** Integration must be treated as a continuous, daily process rather than a discrete final step at the end of a development cycle.

Keeping up with main

SINE NOMINE ASSOCIATES

Branch Integration: The Rebase Strategy

- **The Mechanism:** `git rebase` detaches local commits, updates the branch starting point to match the current mainline, and sequentially replays the local commits on top.
- **The Advantage:** Produces a strictly linear commit history. This simplifies `git log` readability and significantly improves the efficiency of `git bisect` operations.
- **The Disadvantage:** It destroys the original commits and generates entirely new SHA-1 hashes. Rebasing commits that have already been pushed to a shared remote will permanently break the repository state for other developers.

Before rebase:
main: A — B — C
feature: D — E — F

After rebase:
main: A — B — C
feature: \ D' — E' — F' *(replayed on top of main)*

After fast-forward merge:
main: A — B — C — D' — E' — F'

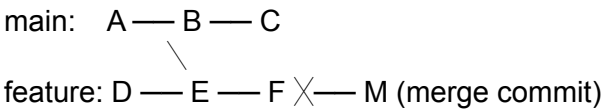
```
git checkout feature
git rebase main       # Replay commits from feature
git checkout main
git merge feature     # Fast-forward (no merge commit needed)
```

Keeping up with main

SINE NOMINE ASSOCIATES

Branch Integration: The Merge Strategy

- **The Mechanism:** `git merge` evaluates two divergent branch histories and creates a new, dedicated commit that combines them.
- **The Advantage:** Provides an exact, unedited historical record. It explicitly documents when a feature diverged and exactly when it was integrated into the mainline.
- **The Disadvantage:** Heavy utilization generates complex, overlapping commit graphs that can complicate visual audits and history tracking.



Result: A linear history with a merge bubble

```
git checkout main
git merge feature
```

Keeping up with main

SINE NOMINE ASSOCIATES

Branch Integration: The Cherry-Pick Strategy

- **The Mechanism:** git cherry-pick extracts a commit from one branch and applies its changes as a *new commit* to the current branch
- **The Advantages:**
  - Allows surgical application of single, critical fixes (e.g., hotfixes) to a stable branch without merging an entire feature or development history.
  - Provides fine-grained control over which changes are propagated across branches.
- **The Disadvantage:**
  - Creates a new, duplicate commit with a new SHA-1 hash, which can make history audit and tracking more difficult.
  - Routine use for integration can lead to a messy, non-linear history and the need to resolve the same conflict multiple times if the original commit is merged later.

main: A — B — C
release: D — E — F

After cherry-pick:
main: A — B — C
release: D — E — F — B`

git checkout release
git cherry-pick B *# Extract and apply a commit from another branch*

Keeping up with main

```
cwills@astro ~/src/gittalk $ git -P log --oneline --graph
* f03c807 (HEAD -> WIP) Fix formatting
* eeb1a19 (main) handle condtion when 2nd parm is zero
* 38cfdd9 Move divide ahead of checks
* f255388 Fix display
* 8fc15b5 Add new function
* 1b22841 Add demo program
* 03c8dc4 Initial project import
cwills@astro ~/src/gittalk $
```

```
cwills@astro ~/src/gittalk2/gittalk $ git -P log --oneline --graph
* 441470a (HEAD -> main) Change int to numeric
* eeb1a19 handle condtion when 2nd parm is zero
* 38cfdd9 (origin/main, origin/HEAD) Move divide ahead of checks
* f255388 Fix display
* 8fc15b5 Add new function
* 1b22841 Add demo program
* 03c8dc4 Initial project import
cwills@astro ~/src/gittalk2/gittalk $
```

Keeping up with main

SINE NOMINE ASSOCIATES

Conflicts during merges, rebasing or cherry-picks

- A commit that is on the branch you are trying to merge, rebase, or cherry-pick conflicts with a commit already on the branch.
- The conflict needs to be resolved by modifying the commit to address the cause of the conflict.

```
#!/usr/bin/python

import sys

n1 = sys.argv[1]
n2 = sys.argv[2]


if n2 != "0":
        print(f"{n1} / {n2} = {int(n1) / int(n2)}")

if n1 > n2:
        print(f"{n1} > {n2}")
else:
        print(f"{n1} < {n2}")
```

```
#!/usr/bin/python

import sys

n1 = sys.argv[1]
n2 = sys.argv[2]


if numeric(n2) != "0":
        print(f"{n1}/{n2} = {numeric(n1) / numeric(n2)}")

if n1 > n2:
        print(f"{n1} > {n2}")
else:
        print(f"{n1} < {n2}")
```

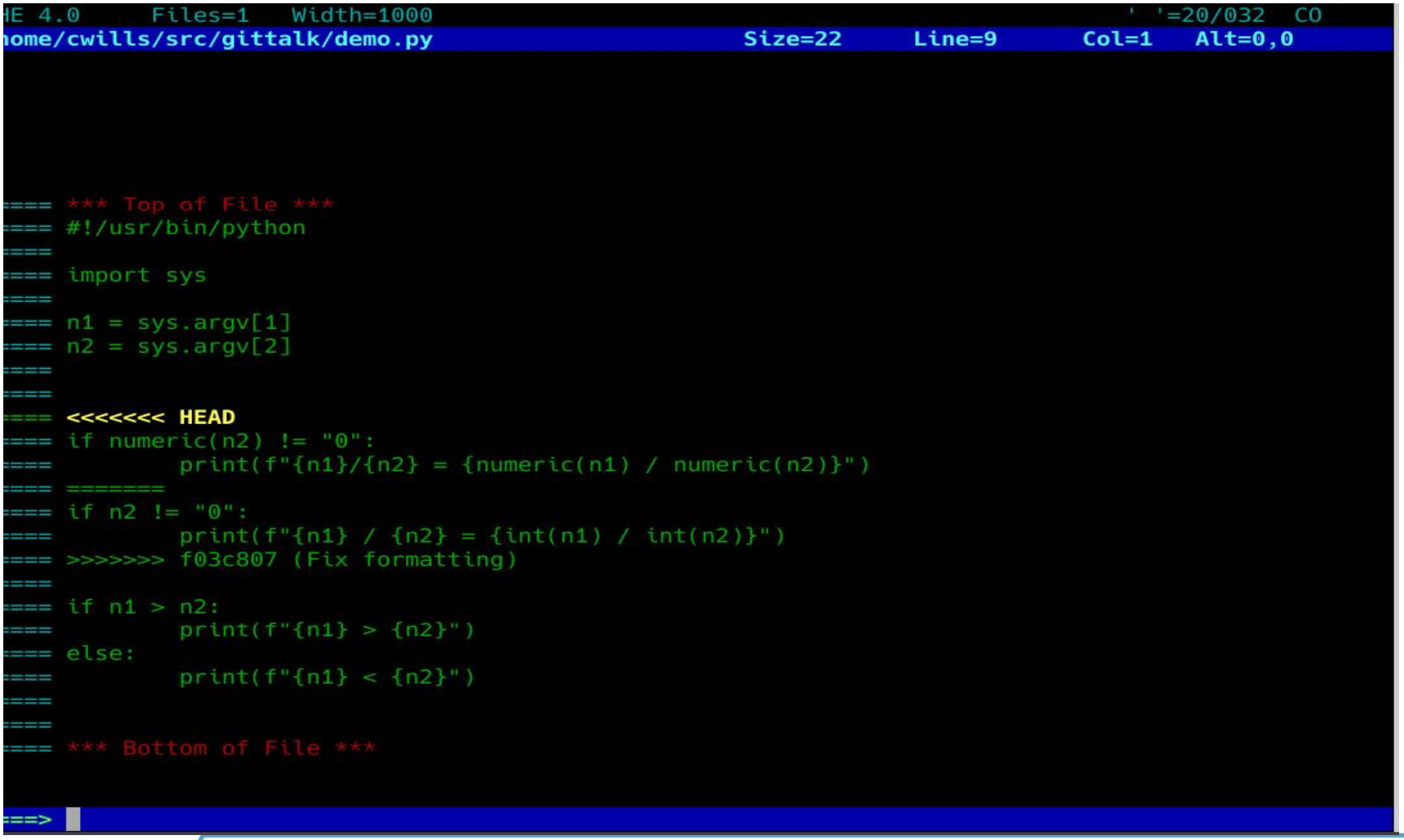
Keeping up with main

SINE NOMINE ASSOCIATES

```
cwills@astro ~/src/gittalk $ git fetch myotherrepo
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 328 bytes | 328.00 KiB/s, done.
From /home/cwills/src/gittalk2/gittalk
   eeb1a19..441470a  main       -> myotherrepo/main
cwills@astro ~/src/gittalk $ git rebase myotherrepo/main
Auto-merging demo.py
CONFLICT (content): Merge conflict in demo.py
error: could not apply f03c807... Fix formatting
hint: Resolve all conflicts manually, mark them as resolved with
hint: "git add/rm <conflicted_files>", then run "git rebase --continue".
hint: You can instead skip this commit: run "git rebase --skip".
hint: To abort and get back to the state before "git rebase", run "git rebase --abort".
hint: Disable this message with "git config set advice.mergeConflict false"
Could not apply f03c807... # Fix formatting
cwills@astro ~/src/gittalk $
```

Keeping up with main

SINE NOMINE ASSOCIATES



Keeping up with main

SINE NOMINE ASSOCIATES

```
cwills@astro ~/src/gittalk $ the demo.py
cwills@astro ~/src/gittalk $ git add demo.py
cwills@astro ~/src/gittalk $ git rebase --continue
[detached HEAD 176658e] Fix formatting
 1 file changed, 3 insertions(+), 3 deletions(-)
Successfully rebased and updated refs/heads/WIP.
cwills@astro ~/src/gittalk $
```

```
Fix formatting

# Conflicts:
#       demo.py

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# interactive rebase in progress; onto 441470a
# Last command done (1 command done):
#    pick f03c807 # Fix formatting
# No commands remaining.
# You are currently rebasing branch 'WIP' on '441470a'.
#
# Changes to be committed:
#       modified:   demo.py
#
~/src/gittalk/.git/COMMIT_EDITMSG" 17L, 431B                    1,1           All
```

Keeping up with main

- **A commit itself is immutable..**
- **but…** the history of a branch isn't..
- **However…** a commit that doesn't have something pointing to it will be cleaned up by Git's garbage collector
- **The Mechanism:** `git rebase -i` interactive rebase allows one to alter a branch by:
  - Use a commit as-is
  - reword a commit message
  - edit a commit
  - squash the commit with the previous commit

```
cwills@astro ~/src/gittalk $ git -P log --oneline
176658e (HEAD -> WIP) Fix formatting
441470a (myotherrepo/main, myotherrepo/HEAD) Change int to numeric
eeb1a19 (main) handle condtion when 2nd parm is zero
38cfdd9 Move divide ahead of checks
f255388 Fix display
8fc15b5 Add new function
1b22841 Add demo program
03c8dc4 Initial project import
cwills@astro ~/src/gittalk $ git rebase -i 441470a
```

Keeping up with main

SINE NOMINE ASSOCIATES

```
pick 176658e # Fix formatting

# Rebase 441470a..176658e onto 441470a (1 command)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup [-C | -c] <commit> = like "squash" but keep only the previous
#                    commit's log message, unless -C is used, in which case
#                    keep only this commit's message; -c is same as -C but
#                    opens the editor
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
#         create a merge commit using the original merge commit's
#         message (or the oneline, if no original merge commit was
#         specified); use -c <commit> to reword the commit message
# u, update-ref <ref> = track a placeholder for the <ref> to be updated
#                       to this position in the new commits. The <ref> is
#                       updated at the end of the rebase
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
~
                                                                1,1           All
```

Keeping up with main

Scary live demo of an interactive rebase

Now where did you say my files are?

SINE NOMINE ASSOCIATES

Content-Addressable Storage (Blobs)

- **The Object Store:** Git does not store tracking data by filename or directory path; everything resides in a key-value database within the `.git/objects` directory.
- **The Blob Object:** Every unique version of a file's data is compressed into a "Binary Large Object" (blob).
- **Cryptographic Addressing:** A blob is identified exclusively by the SHA-1 hash of its byte contents.
- **Inherent Deduplication:** Because addressing is based strictly on content, ten copies of the exact same file across different directories will result in only a single blob being stored on disk.

Now where did you say my files are?

SINE NOMINE ASSOCIATES

Reconstructing the Filesystem (Trees)

- **Metadata Separation:** Blob objects contain strictly file data; they possess zero knowledge of their own filenames, extensions, or execution permissions.
- **The Tree Object:** Git rebuilds the directory structure using "Tree" objects, which function identically to standard UNIX directory listings.
- **Mapping the Data:** A tree object maps human-readable filenames and access modes (e.g., `100644` for standard files, `100755` for executables) to the specific SHA-1 hashes of the underlying blobs.
- **Cheap Renames:** Moving or renaming a file does not alter the file data or create a new blob; it simply updates a few bytes of text within the parent tree object to point to the existing hash.

Swinging back to VM

SINE NOMINE ASSOCIATES

In loose terms

VM Concept	Git Concept
CNTRL file	Branch (a pointer to a state)
Collection of AUX files	Commit Log (the full history)
Atomic change set in the update files	Commit (the single logical change)

Final points...

SINE NOMINE ASSOCIATES

Granular History and Rollbacks

- **Explicit Checkpoints:** You control exactly what goes into the permanent record. You can group related file changes together into a single, logical commit with a descriptive message explaining the "why" behind the code.
- **The Safety Net:** Every single commit is a restorable state. If a regression is found, you can easily revert a specific commit or reset your entire working directory to a known good state from yesterday or last year.
- **Absolute Traceability:** You can instantly query any line of code in any file to see the exact timestamp, the author, and the commit message associated with its last modification.

Final points...

SINE NOMINE ASSOCIATES

Intentional Collaboration

- **Private by Default:** Your local commits and branches are completely private. You can make dozens of messy, incremental commits as you work; no one else sees them until you explicitly decide to share them.
- **Explicit Synchronization:** You choose exactly when to pull updates from the rest of the team and exactly when to push your finished work to the shared repository.
- **Automated Merging:** When you do combine work, Git automatically merges non-overlapping changes. If two developers edit the exact same line, Git stops and explicitly flags the conflict for human review before proceeding.

Q&A

SINE NOMINE ASSOCIATES

Questions?