

Git for everyone

KINDNS: DNS & DNSSEC operational best practices to improve the DNS Ecosystem



These materials are licensed under the Creative Commons Attribution-NonCommercial 4.0 International license (<http://creativecommons.org/licenses/by-nc/4.0/>)



UNIVERSITY OF OREGON



Acknowledgements

- I originally prepared this presentation for an online session during the Dark Ages in December 2021 with funding from ISOC. The original title at the time was *GitHub management for network engineers*. A recording can be found on YouTube as of April 2025:
<https://www.youtube.com/live/fgRjoNiWHJ4?si=ZoqQdA7Oazd6S9HZ>
- Thanks to Aftab Siddiqui and Massimiliano Stucchi for inviting me to put this together and for arranging the funding.
- I keep tinkering with this presentation. It lives on as part of my KINDNS: best practices for DNS operations workshop. Bug reports and suggestions for improvement are welcome.

Philip Paeps
philip@trouble.is
8 April 2025

Share and enjoy!

These materials are licensed under the Creative Commons Attribution-NonCommercial 4.0 International licence

<http://creativecommons.org/licenses/by-nc/4.0/>



Please report bugs, errors or omissions.

Agenda

1. Revision control essentials
2. Git survival kit for DNS ops
3. Using GitHub to collaborate



UNIVERSITY OF OREGON



Computers are better at remembering things than you are.

REVISION CONTROL ESSENTIALS



UNIVERSITY OF OREGON



Revision control for DNS operators

Revision control systems remember changes you make to your DNS. With good revision control hygiene, you can easily:

- Revert configurations to a known working state
- Review changes before deploying them to production
- Recover configurations when servers break
- Collaborate on projects with others without conflicts

Not only for source code and configuration

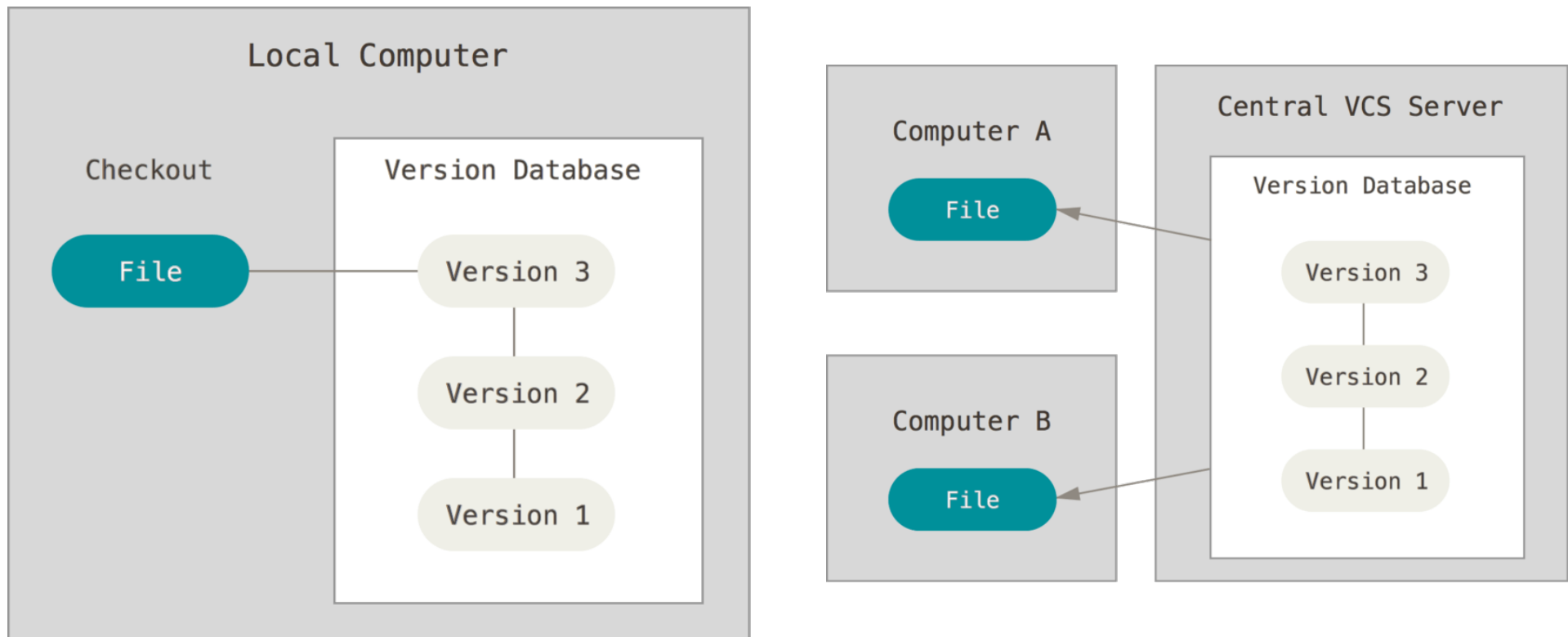
Revision control systems don't care about the data they control. Use them to track changes and collaborate on all sorts of things:

- Internet drafts
- Network policy documents
- Training materials
- Presentations
- DNS configuration files

Revision control options

Amount of control	Pros and cons
Chaos reigns Loose files all over the place	✓ Easy to learn ✗ Impossible to undo changes
Archiving for posterity NFS, SMB, OneDrive, Dropbox,...	✓ Audit and roll back previous versions ✗ Concurrent access nightmares
Revision control CVS, Subversion, Git, etc.	✓ Full control and low-friction collaboration ✗ Learning curve

Basics of revision control



UNIVERSITY OF OREGON

Revision control system? Content addressable filesystem?
Something software people use? A synonym for software people?
Why should DNS operators care?

GIT SURVIVAL KIT FOR DNS OPS



UNIVERSITY OF OREGON



What is Git anyway?

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

From git-scm.com

GitHub is a company providing a cloud service built around **Git**.



<https://xkcd.com/1597/>

Be nice to your future self

The Git **commit** command writes staged changes to the repository. The *commit message* should explain what the changes are intended to do.

The log of a repository are notes to your future self. When things break, you will want to read them.

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

<https://xkcd.com/1296/>

Git commands for everyday use

Get a repository

```
git init  
git clone
```

Manipulate the index

```
git add  
git rm
```

Commit changes

```
git commit
```

Review logs

```
git log  
git show
```

Figure out what's happening

```
git status  
git diff
```

Undo changes

```
git reset  
git checkout
```

Work with others

```
git fetch  
git rebase
```

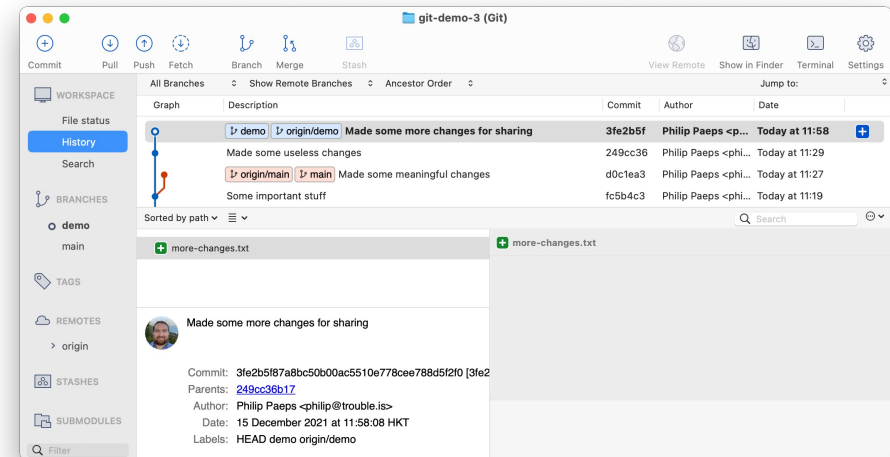


GUI Git tools

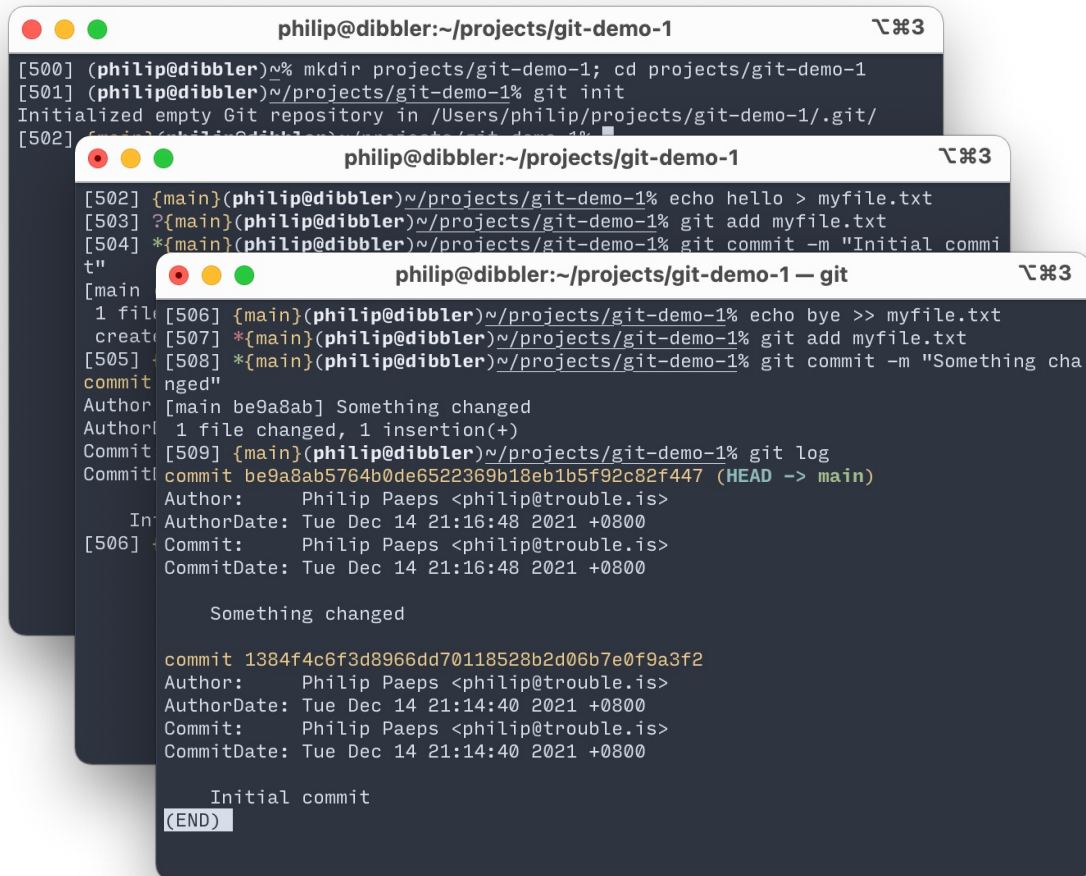
Git comes with two GUIs: **gitk** for browsing branches and **git-gui** for preparing/staging commits. Neither of them is particularly useful.

Atlassian Sourcetree (free) is pretty and works well.

GitHub has desktop clients (also free).



Five-minute intro to Git (demo)



The image shows three overlapping terminal windows from a macOS environment, demonstrating Git workflow steps. The top window shows the creation of a repository. The middle window shows adding a file and committing it. The bottom window shows the commit history.

```
philip@dibbler:~/projects/git-demo-1
[500] (philip@dibbler)~% mkdir projects/git-demo-1; cd projects/git-demo-1
[501] (philip@dibbler)~/projects/git-demo-1% git init
Initialized empty Git repository in /Users/philip/projects/git-demo-1/.git/
[502] (philip@dibbler)~/projects/git-demo-1% echo hello > myfile.txt
[503] ?{main}(philip@dibbler)~/projects/git-demo-1% git add myfile.txt
[504] *{main}(philip@dibbler)~/projects/git-demo-1% git commit -m "Initial commit"
[main] 1 file changed, 1 insertion(+)
create 1 file changed, 1 insertion(+)
[505] *{main}(philip@dibbler)~/projects/git-demo-1% git commit -m "Something changed"
[main be9a8ab] Something changed
Author: Philip Paeps <philip@trouble.is>
Commit: Philip Paeps <philip@trouble.is>
In: AuthorDate: Tue Dec 14 21:16:48 2021 +0800
[506] *{main}(philip@dibbler)~/projects/git-demo-1% git log
commit be9a8ab5764b0de6522369b18eb1b5f92c82f447 (HEAD -> main)
Author: Philip Paeps <philip@trouble.is>
AuthorDate: Tue Dec 14 21:16:48 2021 +0800
Commit: Philip Paeps <philip@trouble.is>
CommitDate: Tue Dec 14 21:16:48 2021 +0800

Something changed

commit 1384f4c6f3d8966dd70118528b2d06b7e0f9a3f2
Author: Philip Paeps <philip@trouble.is>
AuthorDate: Tue Dec 14 21:14:40 2021 +0800
Commit: Philip Paeps <philip@trouble.is>
CommitDate: Tue Dec 14 21:14:40 2021 +0800

Initial commit
(END)
```

Create a new repository

`git init`

Add a file to the staging area

`git add`

Commit changes to the repository

`git commit`

Show history

`git log`



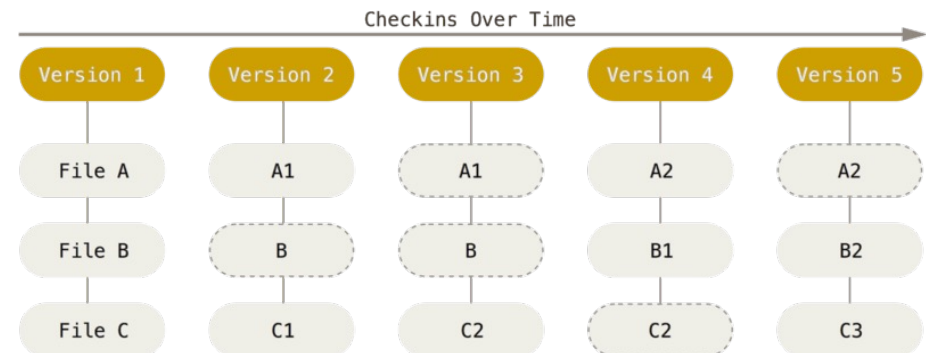
A series of snapshots

Each **commit** is a snapshot of the repository at that point in time.

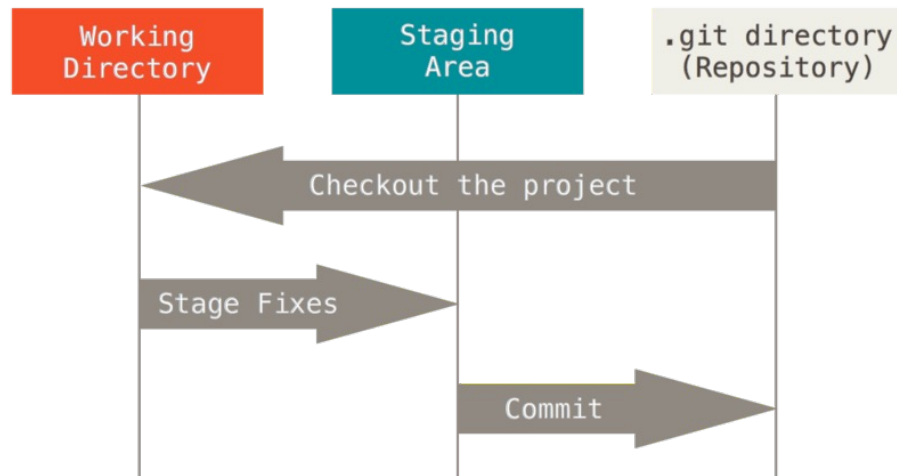
Git references snapshots by the SHA-1 **hash** of their contents.

Most Git operations are **local**.

Git generally only **adds** data. It is difficult to *lose data* once committed.



Git terminology: states and the index



Three main states of Git:

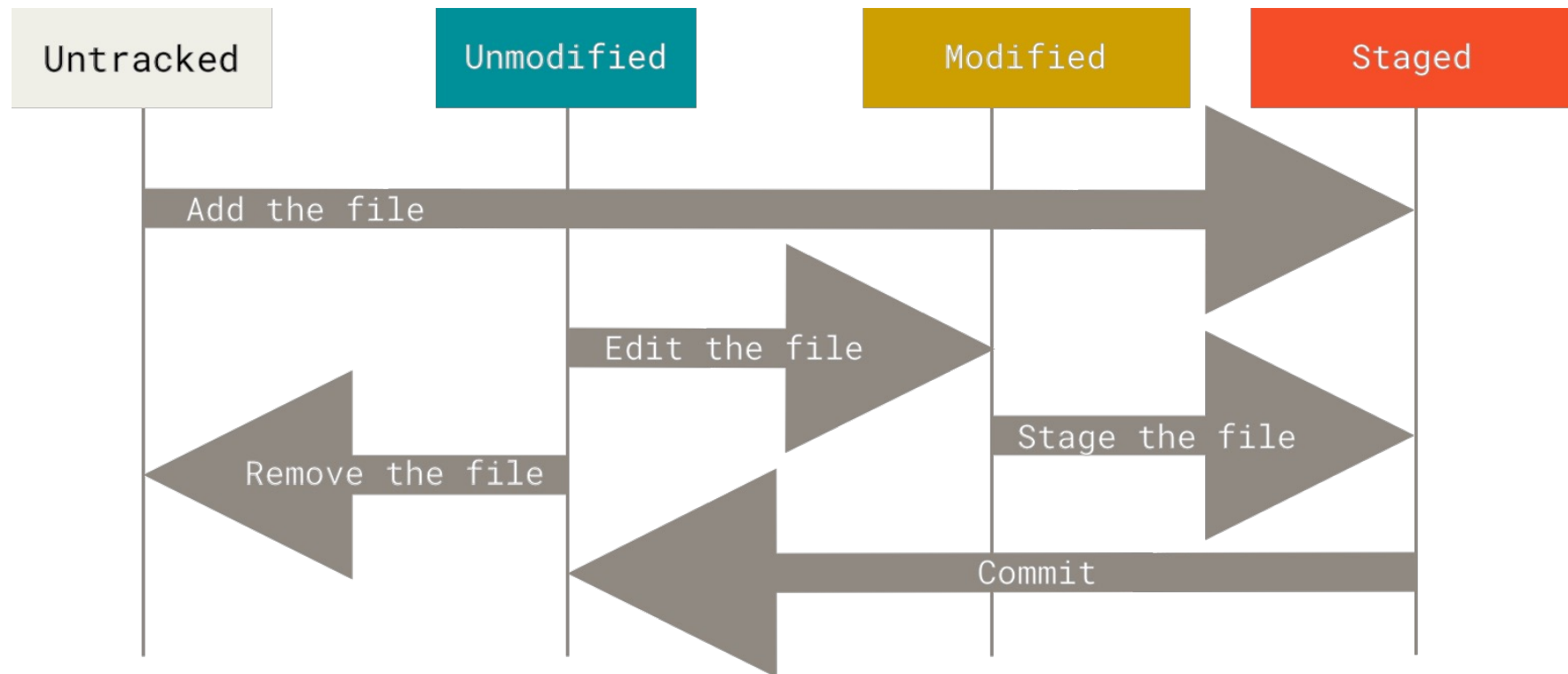
- **Modified** files have uncommitted changes
- **Staged** changes will be written to the repository in the next commit ("index")
- **Committed** changes are safely stored

Not really a state:

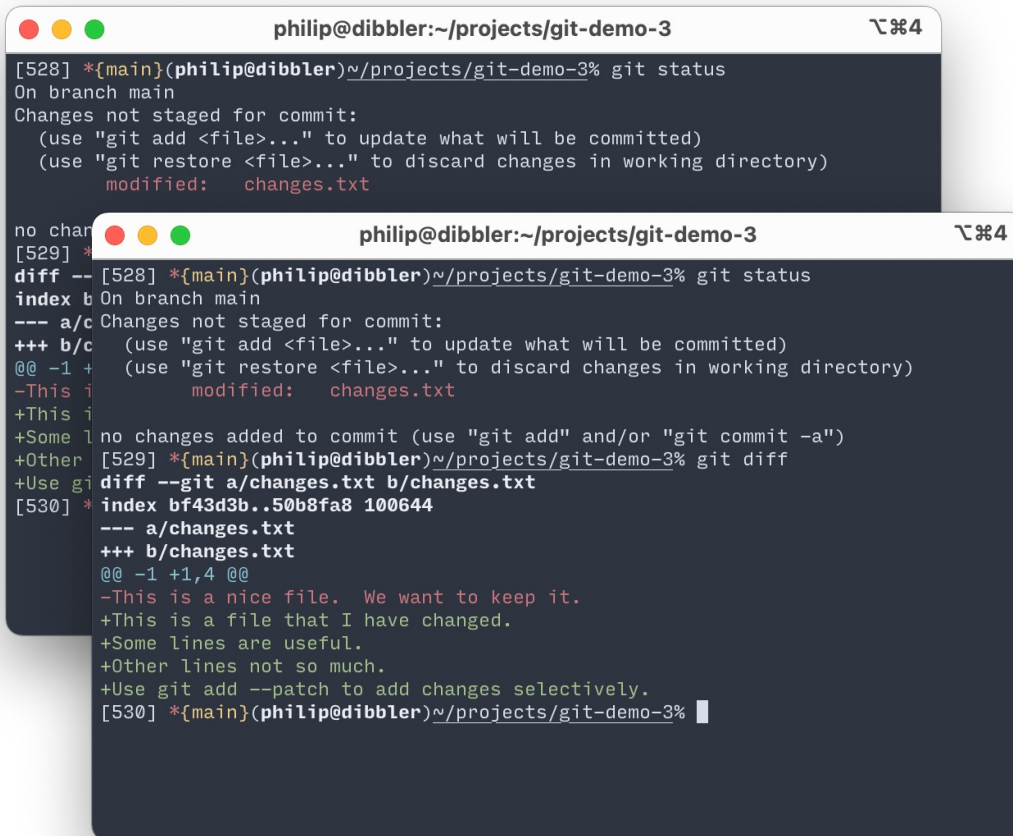
- **Untracked** files are unknown to Git



Git workflow: recording changes



Using the index effectively (demo)



```
philip@dibbler:~/projects/git-demo-3
[528] *(main)(philip@dibbler)~/projects/git-demo-3% git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   changes.txt

no changes added to commit (use "git add" and/or "git commit -a")
[529] *(main)(philip@dibbler)~/projects/git-demo-3% git diff
diff --git a/changes.txt b/changes.txt
index bf43d3b..50b8fa8 100644
--- a/changes.txt
+++ b/changes.txt
@@ -1,4 @@
-This is a nice file. We want to keep it.
+This is a file that I have changed.
+Some lines are useful.
+Other lines not so much.
+Use git add --patch to add changes selectively.
[530] *(main)(philip@dibbler)~/projects/git-demo-3%
```

Stage changes before committing

`git add --patch`

Undo local changes

`git restore`

Keeping track of local changes

`git status`

`git diff`

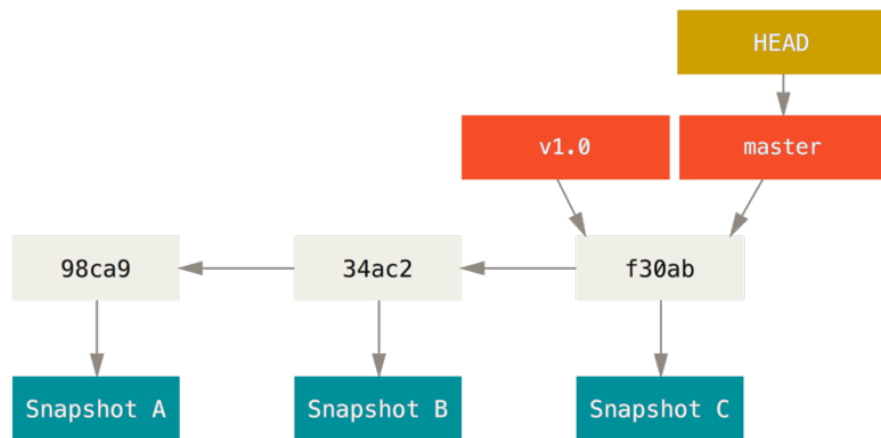


Basics of Git branches

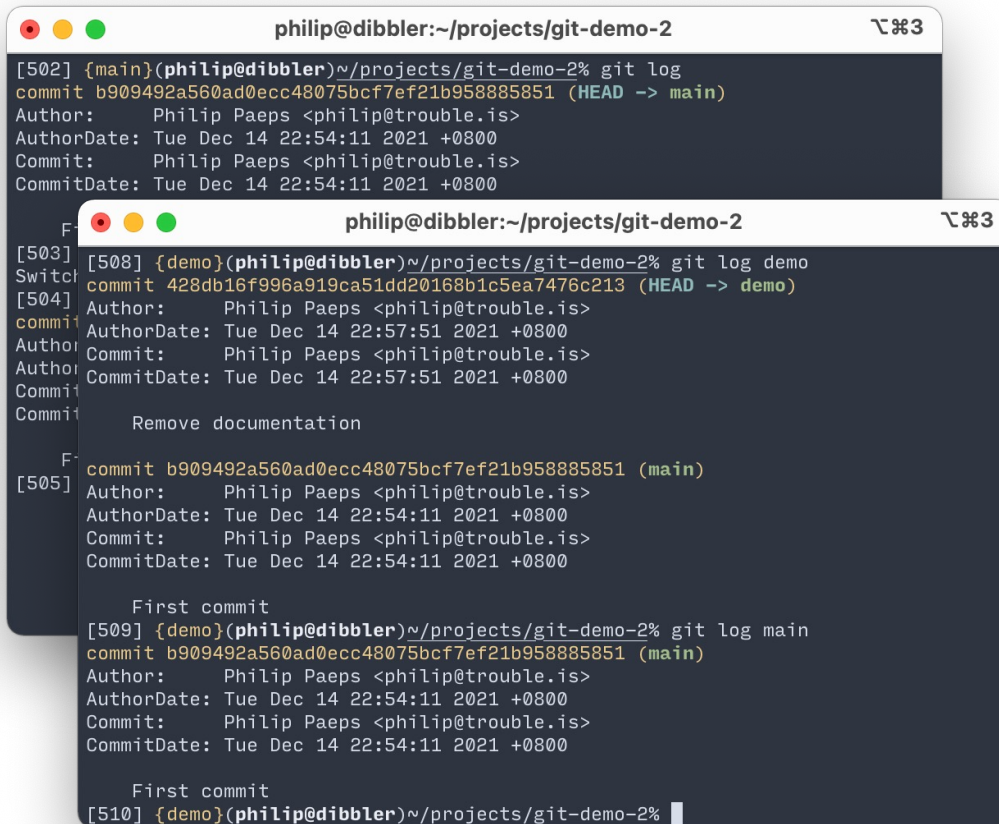
A branch is a named pointer to a snapshot (commit) known to Git.

Git makes it easy to switch between branches and record distinct histories.

The HEAD points to the currently checked out branch (commit).



Branching essentials (demo)



```
philip@dibbler:~/projects/git-demo-2
[502] {main}(philip@dibbler)~/projects/git-demo-2% git log
commit b909492a560ad0ecc48075bcf7ef21b958885851 (HEAD -> main)
Author: Philip Paeps <philip@trouble.is>
AuthorDate: Tue Dec 14 22:54:11 2021 +0800
Commit: Philip Paeps <philip@trouble.is>
CommitDate: Tue Dec 14 22:54:11 2021 +0800

F
philip@dibbler:~/projects/git-demo-2
[503] {demo}(philip@dibbler)~/projects/git-demo-2% git log demo
commit 428db16f996a919ca51dd20168b1c5ea7476c213 (HEAD -> demo)
Author: Philip Paeps <philip@trouble.is>
AuthorDate: Tue Dec 14 22:57:51 2021 +0800
Commit: Philip Paeps <philip@trouble.is>
CommitDate: Tue Dec 14 22:57:51 2021 +0800
Remove documentation

F
philip@dibbler:~/projects/git-demo-2
[504] {demo}(philip@dibbler)~/projects/git-demo-2% git log main
commit b909492a560ad0ecc48075bcf7ef21b958885851 (main)
Author: Philip Paeps <philip@trouble.is>
AuthorDate: Tue Dec 14 22:54:11 2021 +0800
Commit: Philip Paeps <philip@trouble.is>
CommitDate: Tue Dec 14 22:54:11 2021 +0800
First commit

F
philip@dibbler:~/projects/git-demo-2
[505] {demo}(philip@dibbler)~/projects/git-demo-2% git log demo
commit b909492a560ad0ecc48075bcf7ef21b958885851 (main)
Author: Philip Paeps <philip@trouble.is>
AuthorDate: Tue Dec 14 22:54:11 2021 +0800
Commit: Philip Paeps <philip@trouble.is>
CommitDate: Tue Dec 14 22:54:11 2021 +0800
First commit

F
philip@dibbler:~/projects/git-demo-2
[509] {demo}(philip@dibbler)~/projects/git-demo-2% git log main
commit b909492a560ad0ecc48075bcf7ef21b958885851 (main)
Author: Philip Paeps <philip@trouble.is>
AuthorDate: Tue Dec 14 22:54:11 2021 +0800
Commit: Philip Paeps <philip@trouble.is>
CommitDate: Tue Dec 14 22:54:11 2021 +0800
First commit

F
philip@dibbler:~/projects/git-demo-2
[510] {demo}(philip@dibbler)~/projects/git-demo-2%
```

Create a new branch

```
git branch <branch>
git checkout -b <branch>
```

Switching between branches

```
git checkout <branch>
```

Keeping track of changes on branches

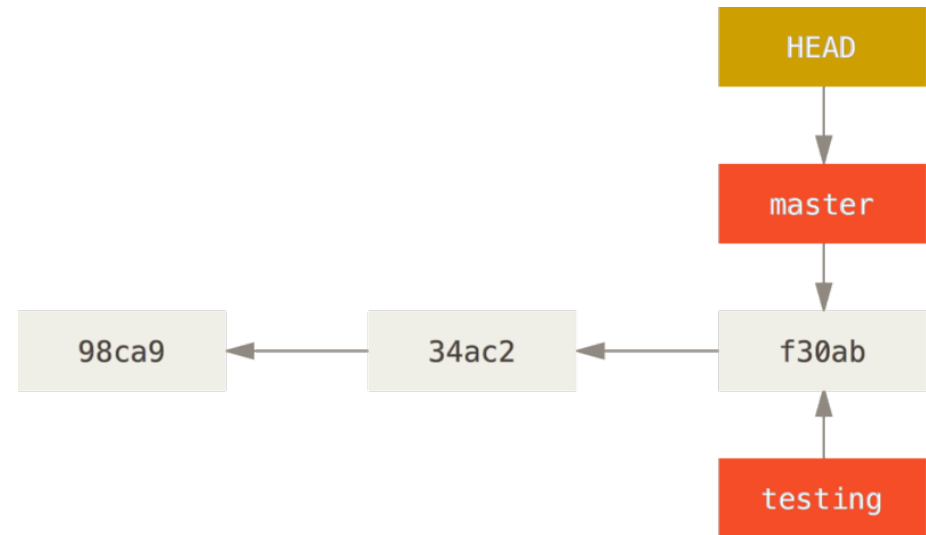
```
git log <branch>
git diff <branch>
```



Branches: creating a branch

Creating a branch adds a new pointer. The HEAD does not move.

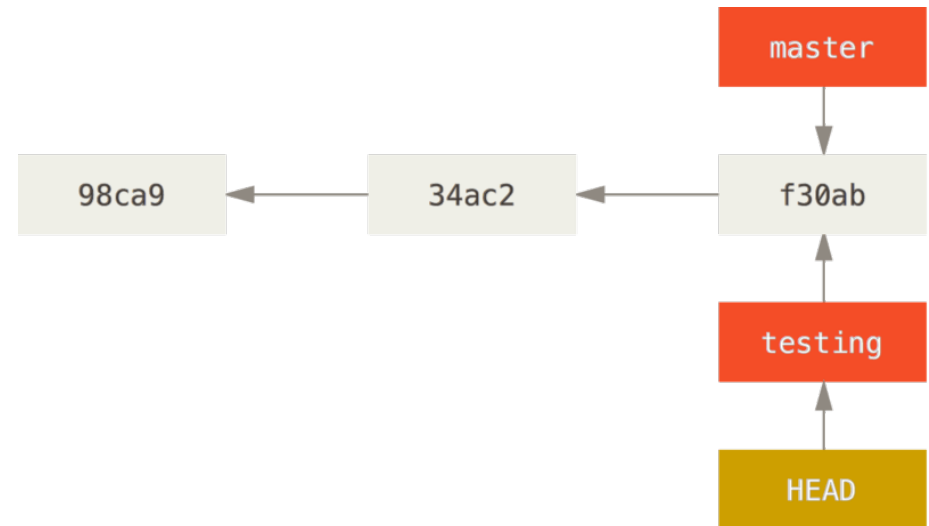
```
git branch testing
```



Branches: switching to another branch (1)

Switching to a branch moves the HEAD.

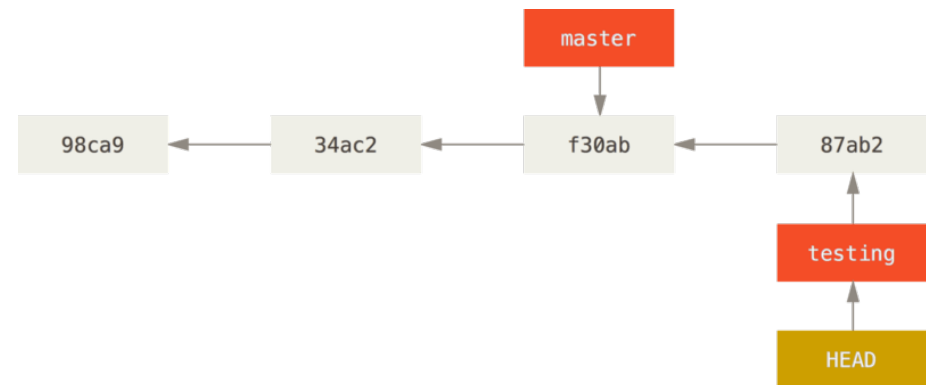
```
git checkout testing
```



Branches: committing to a branch

Committing a change moves the current branch and the HEAD.

```
$EDITOR file.txt  
git commit -m "change  
made"
```

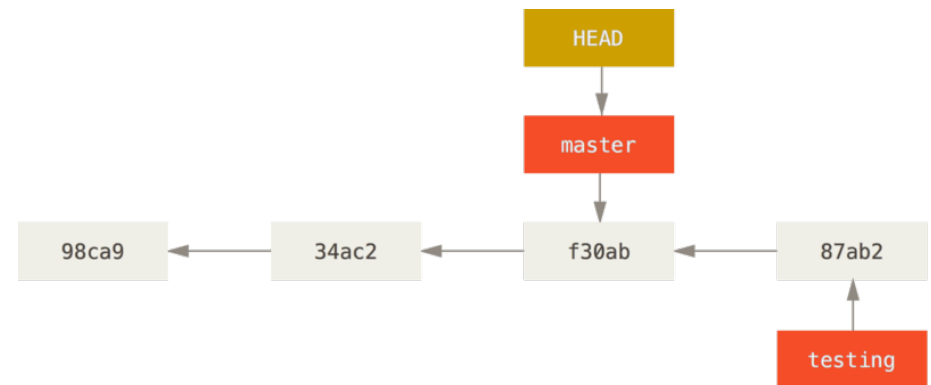


Branches: switching to another branch (2)

Switching to a branch moves the HEAD.

```
git checkout testing
```

The commit only exists on the testing branch.

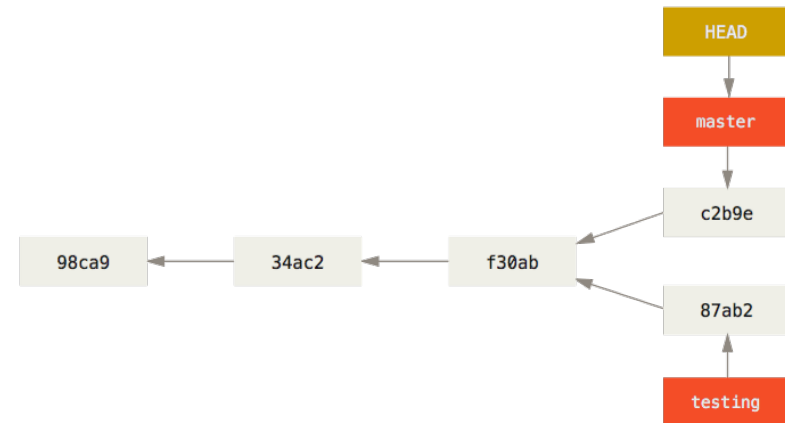


Branches: divergent histories

Committing a change moves the current branch and the HEAD.

```
$EDITOR file.txt  
git commit -m "change  
made"
```

The histories have diverged.
Switching between master and
testing will show their respective
histories.



Using branches to track changes (demo)

```
philip@dibbler:~/projects/git-demo-3 — git
[554] {demo}(philip@dibbler)~/projects/git-demo-3% git log
commit 249cc36b17f2494063d07b4a630208cc60bd6cec (HEAD -> demo)
Author: Philip Paeps <philip@trouble.is>
AuthorDate: Wed Dec 15 11:29:28 2021 +0800
Commit: Philip Paeps <philip@trouble.is>
CommitDate: Wed Dec 15 11:29:28 2021 +0800

Ma

philip@dibbler:~/projects/git-demo-3
[577] {demo}(philip@dibbler)~/projects/git-demo-3% git log --graph --oneline
commit * d0c1ea3 (HEAD -> demo, main) Made some meaningful changes
Author * fc5b4c3 Some important stuff
Author * d27a128 First commit
Commit [578] {demo}(philip@dibbler)~/projects/git-demo-3% git reflog
Commit d0c1ea3 (HEAD -> demo, main) HEAD@{0}: reset: moving to d0c1ea3
Commit d0c1ea3 (HEAD -> demo, main) HEAD@{1}: reset: moving to main
Sc 249cc36 HEAD@{2}: commit: Made some useless changes
fc5b4c3 HEAD@{3}: reset: moving to main^
commit d0c1ea3 (HEAD -> demo, main) HEAD@{4}: checkout: moving from main to demo
Author d0c1ea3 (HEAD -> demo, main) HEAD@{5}: commit: Made some meaningful changes
Author fc5b4c3 HEAD@{6}: commit: Some important stuff
Commit d27a128 HEAD@{7}: commit (initial): First commit
Commit [579] {demo}(philip@dibbler)~/projects/git-demo-3% git reset 249cc36
[580] {demo}(philip@dibbler)~/projects/git-demo-3% git log --graph --oneline
Fr * 249cc36 (HEAD -> demo) Made some useless changes
(EN) * fc5b4c3 Some important stuff
* d27a128 First commit
[581] {demo}(philip@dibbler)~/projects/git-demo-3% █
```

Remembering where you've been

`git reflog`

Moving branches

`git reset`

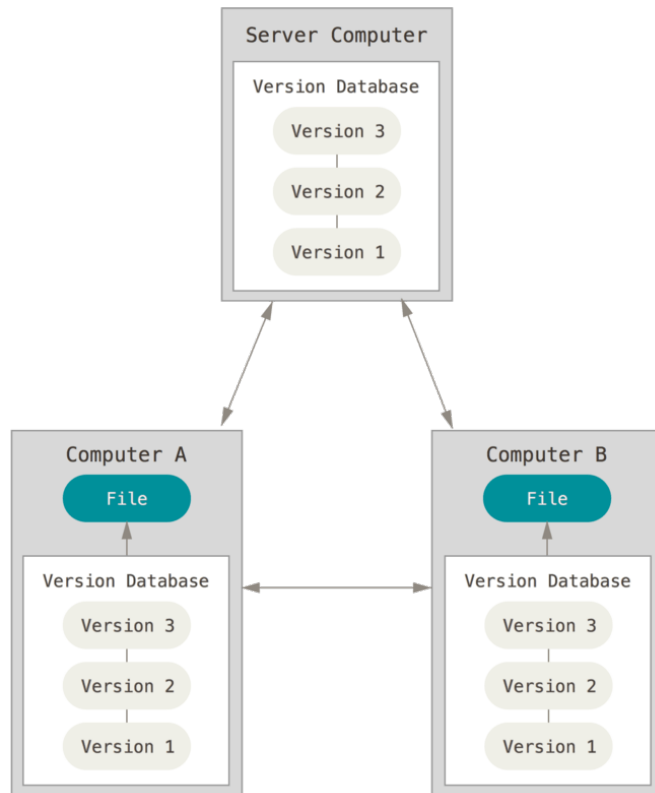
Keeping track of changes on branches

`git log --graph <branch>`

`git diff <branch>`



Remote repositories



Git is a **distributed** revision control system. Adding **remote** repositories enables sharing changes with others.

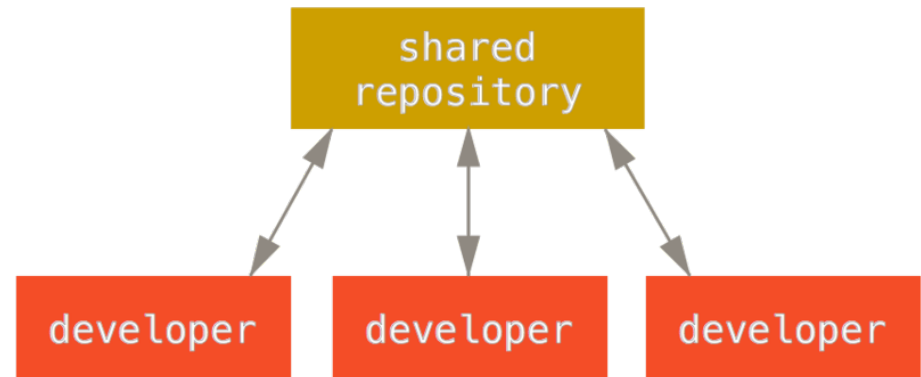
Notes that “remote” repositories can be elsewhere on the “local” machine too.



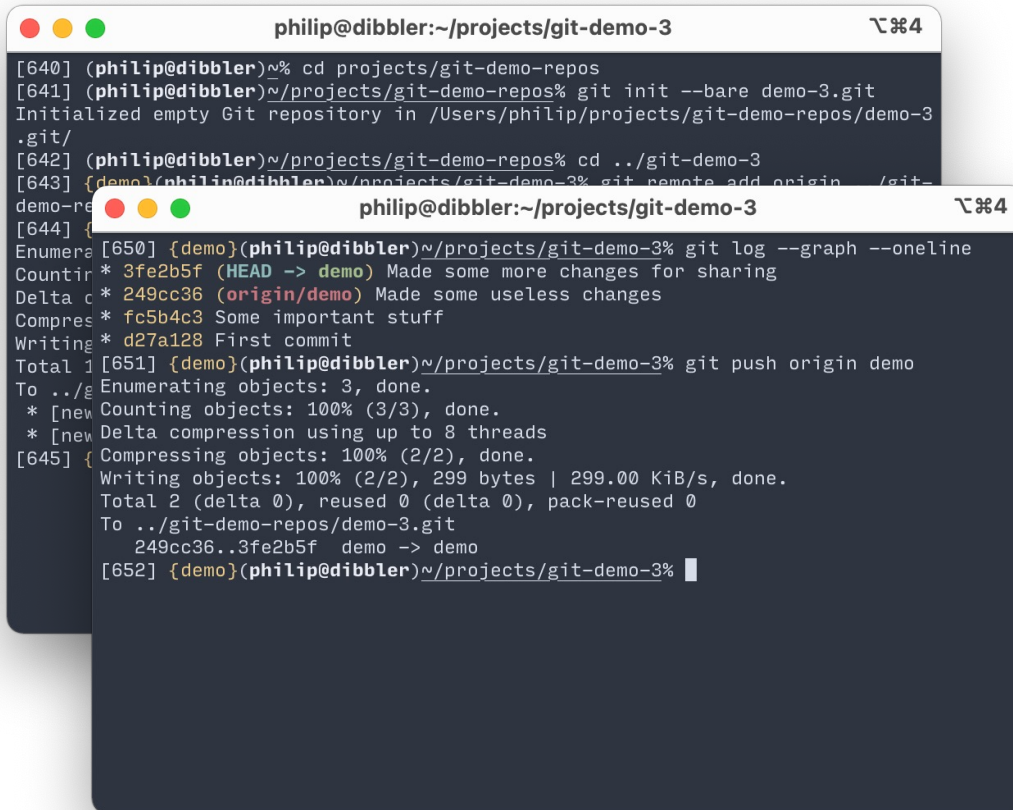
Working with repositories

A remote is a complete **clone** of the repository including all history. This makes collaborating with others easy.

There are several possible workflows of differing complexity. Most of these are irrelevant to network engineers.



Using remote repositories (demo)



The image shows two overlapping terminal windows. The top window, titled 'philip@dibbler:~/projects/git-demo-3', shows the following commands and output:

```
[640] (philip@dibbler)~% cd projects/git-demo-repos
[641] (philip@dibbler)~/projects/git-demo-repos% git init --bare demo-3.git
Initialized empty Git repository in /Users/philip/projects/git-demo-repos/demo-3.git/
[642] (philip@dibbler)~/projects/git-demo-repos% cd ../git-demo-3
[643] {demo}(philip@dibbler)~/projects/git-demo-3% git remote add origin /git-demo-repos/demo-3.git
[644] {demo}(philip@dibbler)~/projects/git-demo-3% git log --graph --oneline
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 299 bytes | 299.00 KiB/s, done.
Total 2 (delta 0), reused 0 (delta 0), pack-reused 0
To ../git-demo-repos/demo-3.git
249cc36..3fe2b5f demo -> demo
[652] {demo}(philip@dibbler)~/projects/git-demo-3%
```

The bottom window, also titled 'philip@dibbler:~/projects/git-demo-3', shows the following commands and output:

```
[650] {demo}(philip@dibbler)~/projects/git-demo-3% git log --graph --oneline
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 299 bytes | 299.00 KiB/s, done.
Total 2 (delta 0), reused 0 (delta 0), pack-reused 0
To ../git-demo-repos/demo-3.git
249cc36..3fe2b5f demo -> demo
[651] {demo}(philip@dibbler)~/projects/git-demo-3% git push origin demo
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 299 bytes | 299.00 KiB/s, done.
Total 2 (delta 0), reused 0 (delta 0), pack-reused 0
To ../git-demo-repos/demo-3.git
249cc36..3fe2b5f demo -> demo
[652] {demo}(philip@dibbler)~/projects/git-demo-3%
```

Adding remote repositories

`git remote add <name> <URL>`

Sharing changes with remotes

`git push <remote> <branch>`

Getting changes from others

`git fetch <remote>`

`git fetch --all`

Merging changes from others

`git rebase <branch>`



Collaboration tools and Git repository hosting.

GITHUB

Tools for collaboration

GitHub provides hosting for Git repositories.

Superficially targeted at software projects but great for any Git repository.

Issue tracker. Pull requests. Wiki.

GitHub



UNIVERSITY OF OREGON



The GitHub workflow

1. Fork a repository from a project
2. Clone your fork and make changes on a branch
3. Push the branch to your namespace
4. Create a Pull Request in the project repository
5. Discuss changes and push updates to your branch
6. Project owner merges the accepted pull request

Using GitHub for automation

GitHub has nice integrations
with continuous deployment /
continuous integration
workflows.

Credits and further reading

Most of the images in this presentation are from the excellent “Pro Git” book by Scott Chacon and Ben Straub. (CC BY-NC-SA 3.0)

Book: <https://git-scm.com/book/en/v2/>

Source code: <https://github.com/progit/progit2>

GitHub cheat sheet

<https://training.github.com/downloads/github-git-cheat-sheet/>

Escaping a Git mess (Justin Hileman)

<http://justinhileman.info/article/git-pretty/>

