Intro to Git

Kevin Adler
kadler@us.ibm.com

© 2015 IBM Corporation

What is it?

"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." - git-scm.com

© 2016 IBM Corporation

What is it really?

- Version Control System (VCS) or Software Configuration Management System (SCM)
  - track history of changes to your source code
  - multiple branches (dev/prod, v7/v8/v9, …)
- Similar products
  - Subversion
  - CVS
  - Perforce
  - Visual Source Safe
  - ClearCase

© 2016 IBM Corporation

Distributed?

- no distinction between “client” and server
  - no “master” repository
  - every checkout is a peer repository
  - every checkout has the “full” history
  - able to do most things disconnected
- Other DVCS products
  - Darcs
  - BitKeeper
  - Mercurial
  - Bazaar

© 2016 IBM Corporation

History Lesson

- Linux developers used to use BitKeeper, proprietary DVCS
- BitKeeper gave free licenses to Linux developers
- Some developers didn't want to use proprietary BitKeeper software, reverse engineered the client protocol
- BitKeeper revoked free licenses due to reverse engineering
- Linus Torvalds surveyed available projects and found all lacking, decides to write his own
- After roughly 4 days of development, git is announced (Apr 6, 2005) and the next day is self-hosting
- Linus used git to manage the 2.6.12 Linux release (Jun 18, 2005)
- Development handed over to Junio Hamano shortly thereafter
- git 1.0 release occurs December 2005

© 2016 IBM Corporation

How to Get Git

- Perzl RPMs
- Use ibmichroot scripts to install (bitbucket or OPS Option 3)
- Old version (1.8)
- SSH, HTTPS, FTP, GIT support
- Perl support

- 5733-OPS Option 6
- Need PTF SI61060 or superseding
- Newer version (2.8)
- *Only SSH and GIT supported
- *No Perl support

* Support coming in the future

© 2016 IBM Corporation

Git design

- Git stores entire files, not differences
  - better speed when traversing history
  - uses compression to save space
  - more like a mini filesystem, less like a VCS
- Lots of checksums
  - Git uses SHA-1 checksums on data
  - Objects are referred to by checksum
    - blob – source code, text data, image, ...
    - tree – pointers to blob or sub-tree
    - commit – pointer to tree, pointer to parent commit, commit metadata
    - tag – named pointer to a git commit (v1.2.3, known_good_state, …)
  - corruption is detectable due to checksums

© 2016 IBM Corporation

Git areas and states

- Working directory
  - full copy of all the source code in your project
  - where you make local changes
  - affected by clone, reset, checkout, merge, pull operations
- Staging Area
  - stores changes ready to be committed
  - also called the index (.git/index)
  - affected by reset operation
- Repository (.git database)
  - stores files that have been committed
  - most operations undo-able
  - affected by commit, reset, fetch, branch, push (remote)

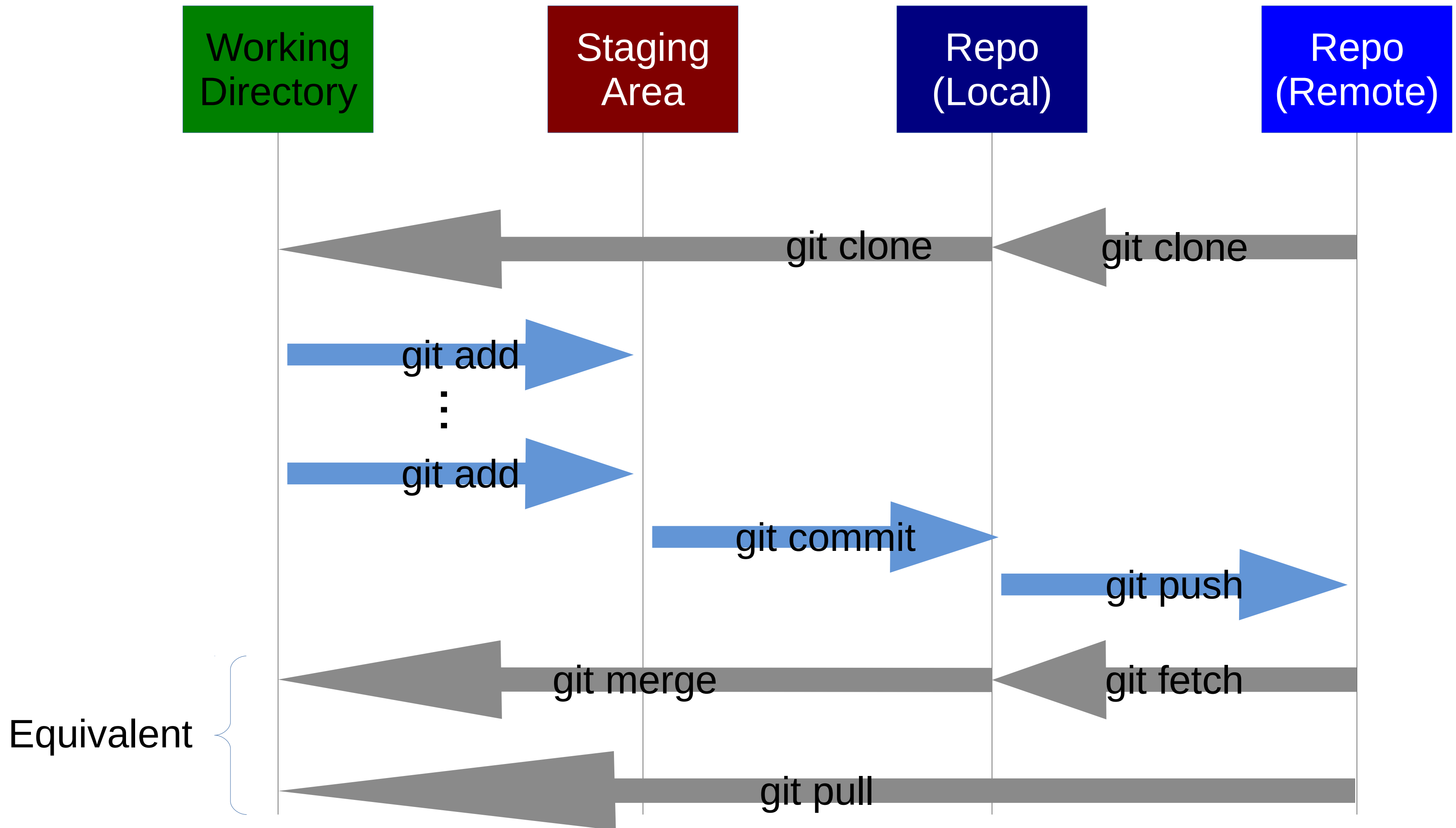
© 2016 IBM Corporation

Basic Workflow

1) Create local git repository
   - `git clone url://mygitrepo.git or git init .`
   - `git supports many url types: ftp, http, ssh, ...`
2) Edit files in working directory
3) Add changed files to staging area
   - `git add foo.py bar.js baz.rb`
4) Commit staging area to repository
   - `git commit -m 'Commit message'`
5) (Optional) Push history to remote repository
   - `git push`
6) (Optional) Pull history from remote repository
   - `git pull`

© 2016 IBM Corporation

Workflow Diagram



© 2016 IBM Corporation

First time setup

- Git has many knobs and buttons
- Configuration settings can be
  - global (~/.gitconfig)
  - per-repo (repo/.git/config)
- Syntax is pretty simple, but best to use `git config`

```
$ git config --global user.name "Kevin Adler"
$ git config --global user.email "kadler@us.ibm.com"

$ git config --list # show all configuration settings
$ git config user.name # show configuration setting
```

© 2016 IBM Corporation

Creating a project

- To start using git, you must create a local git repository
- If you don't, git will give you an error:

```
$ git status

fatal: Not a git repository (or any of the parent directories): .git
```

- Two options
  - Clone an existing repo
  - Initialize a new repo

© 2016 IBM Corporation

Initializing a git repository

```
$ git init ~/my_project # creates directory if needed

Initialized empty Git repository in
/home/kadler/my_project/.git/

$ cd ~/my_project


$ ls -a

. .. .git

$ ls .git

branches  config  description  HEAD  hooks  info
objects  refs
```

© 2016 IBM Corporation

Making some changes

```
$ touch foo bar baz

$ git status

On branch master

Initial commit


Untracked files:

  (use "git add <file>..." to include in what will be committed)

        bar

        baz

        foo

nothing added to commit but untracked files present (use "git add" to track)
```

© 2016 IBM Corporation

Staging the changes

```
$ git add foo bar baz
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   bar
        new file:   baz
        new file:   foo
```

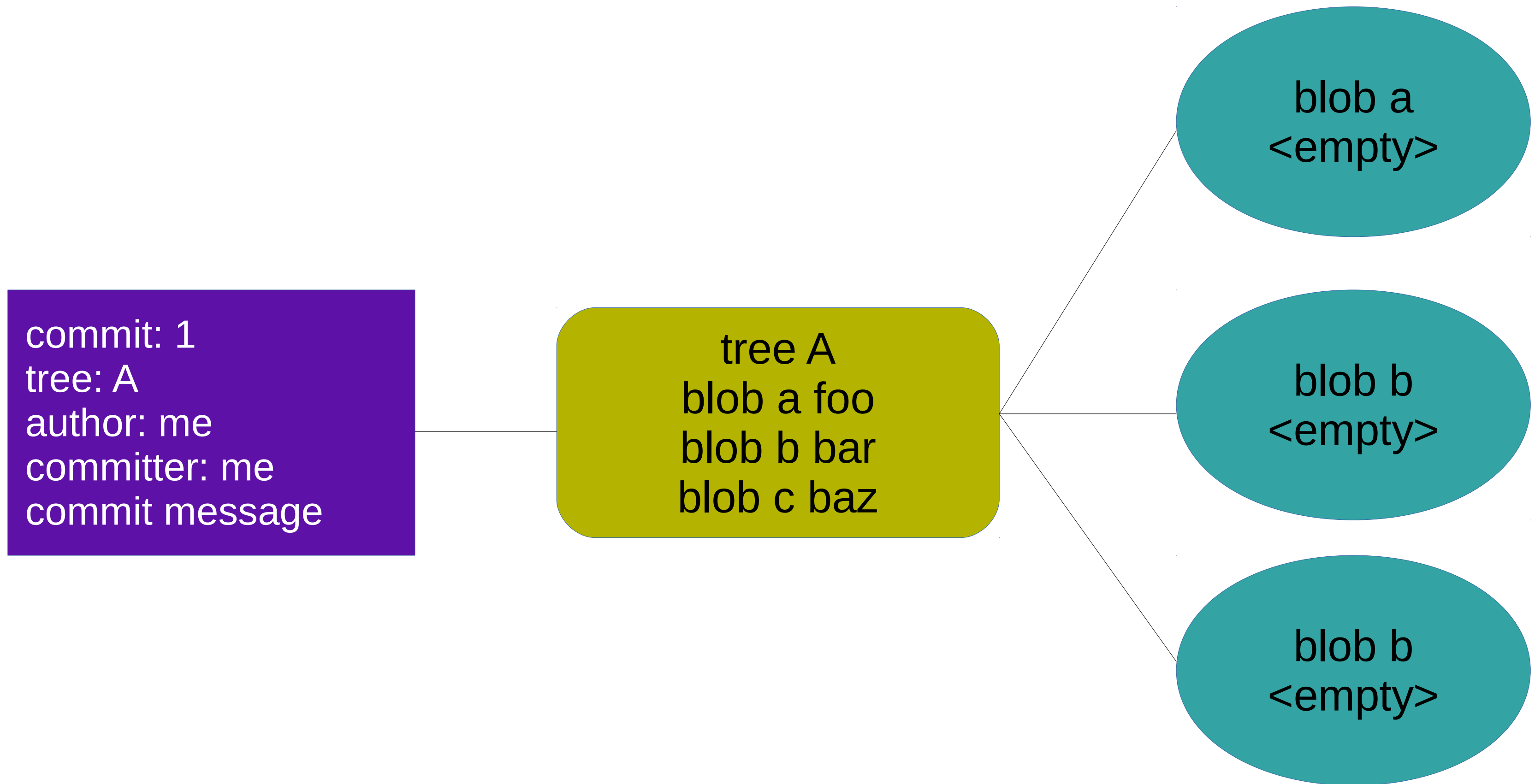
© 2016 IBM Corporation

Committing the changes

```
$ git commit -m "Add foo, bar, and baz”
[master (root-commit) e287cdc] Add foo, bar, and baz
 3 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 bar
 create mode 100644 baz
 create mode 100644 foo
$ git log
commit e287cdc798bc8c01742b8f562c4b3a7255d1884f
Author: Kevin Adler <kadler@us.ibm.com>
Date:   Tue Aug 2 18:11:59 2016 -0500

    Add foo, bar, and baz
```

© 2016 IBM Corporation

Behind the scenes



© 2016 IBM Corporation

Adding some data

```
$ echo 'My name is Bar' > bar

$ git status

On branch master

Changes not staged for commit:

  (use "git add <file>..." to update what will be committed)

  (use "git checkout -- <file>..." to discard changes in
working directory)


        modified:   bar


no changes added to commit (use "git add" and/or "git commit
-a")

$ git commit -a -m 'Add some test data to bar'
```

© 2016 IBM Corporation

Seeing differences

```
$ echo 'My name is Baz' > baz
$ git diff
diff --git a/baz b/baz
index e69de29..5f08a6f 100644
--- a/baz
+++ b/baz
@@ -0,0 +1 @@
+My name is Baz
```

© 2016 IBM Corporation

Seeing staged differences

```
$ git add baz
$ git diff
# no differences
$ git diff --cached
diff --git a/baz b/baz
index e69de29..5f08a6f 100644
--- a/baz
+++ b/baz
@@ -0,0 +1 @@
+My name is Baz
```

© 2016 IBM Corporation

Seeing staged differences

```
$ echo "My name is baz" > baz

$ git diff

diff --git a/baz b/baz

index 5f08a6f..e52c00b 100644

--- a/baz

+++ b/baz

@@ -1 +1,2 @@

-My name is Baz

+My name is baz
```

© 2016 IBM Corporation

Seeing staged differences

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   baz

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   baz
```

© 2016 IBM Corporation

Undoing changes in your working directory

```
$ git checkout -- baz

$ git diff

# no changes

$ git diff --cached

diff --git a/baz b/baz

index e69de29..5f08a6f 100644

--- a/baz

+++ b/baz

@@ -0,0 +1 @@

+My name is Baz
```

© 2016 IBM Corporation

Unstaging changes

```
$ git reset HEAD baz

$ git diff

diff --git a/baz b/baz

index e69de29..5f08a6f 100644

--- a/baz

+++ b/baz

@@ -0,0 +1 @@

+My name is Baz


$ git checkout -- baz
```

© 2016 IBM Corporation

What's this HEAD business?

- HEAD is a special named commit
- Always points to the commit that is currently checked out
- HEAD changes ...
  - when you make a new commit
  - when you checkout a branch, tag, or commit
- You are always at HEAD

© 2016 IBM Corporation

Removing files

```
$ git rm baz
rm 'baz'
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    baz
$ git commit -m "Don't need baz"
```

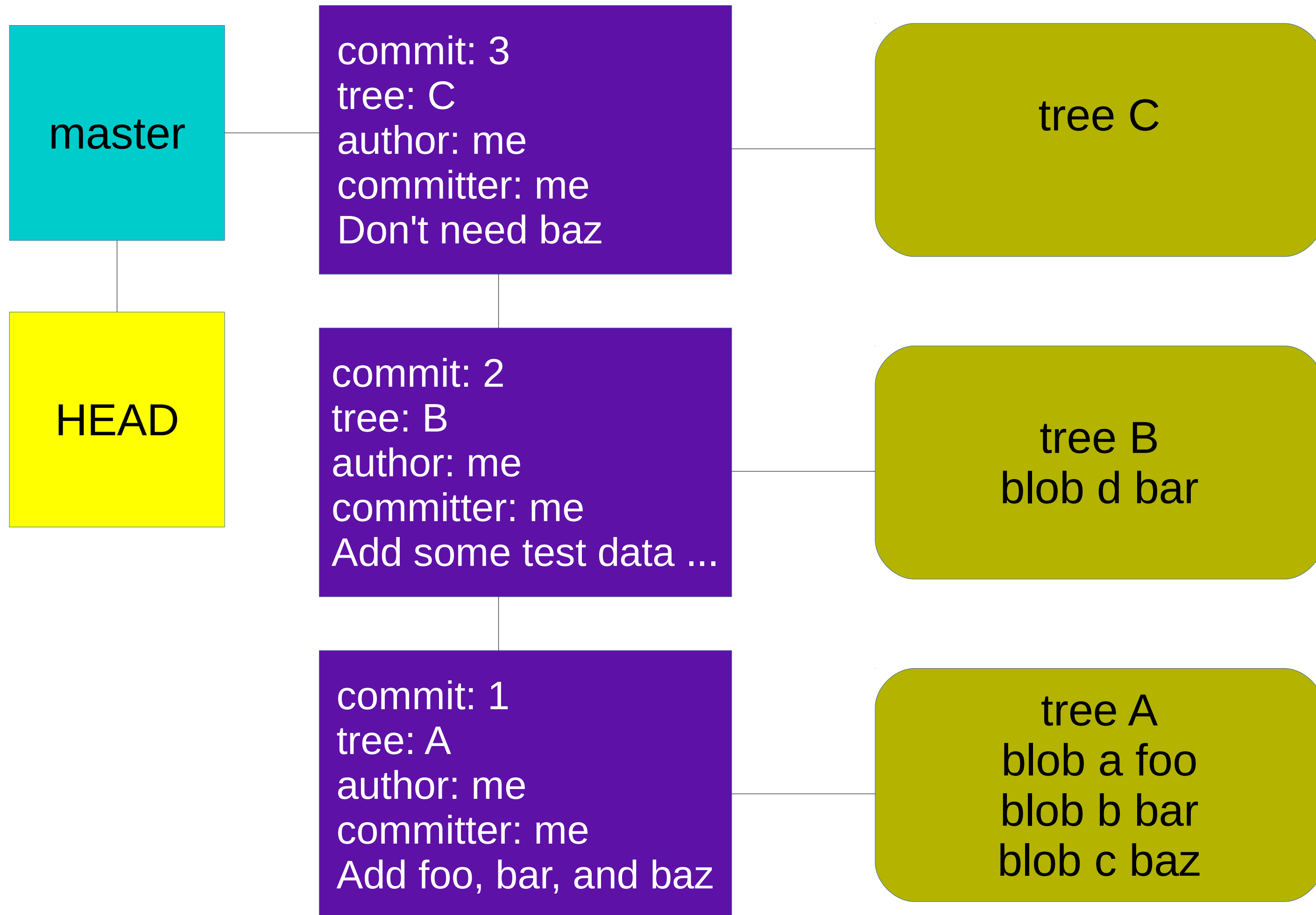
© 2016 IBM Corporation

Viewing the log

```
$ git log
commit 3d1f069131b14be024ddd1752e683c3c7d2e9c59
Author: Kevin Adler <kadler@us.ibm.com>
Date:   Wed Aug 3 17:01:54 2016 -0500
    Don't need baz
commit c947b45e7c1ab317d1fa9805605afbf7c5aaf118
Author: Kevin Adler <kadler@us.ibm.com>
Date:   Wed Aug 3 16:51:33 2016 -0500
    Add some test data to bar
commit e287cdc798bc8c01742b8f562c4b3a7255d1884f
Author: Kevin Adler <kadler@us.ibm.com>
Date:   Tue Aug 2 18:11:59 2016 -0500
    Add foo, bar, and baz
```

© 2016 IBM Corporation

Behind the scenes



© 2016 IBM Corporation

Fancier Log

```
$ git log --graph --decorate --pretty=oneline
--abbrev-commit
```

```
* 3d1f069 (HEAD, master) Don't need baz
* c947b45 Add some test data to bar
* e287cdc Add foo, bar, and baz
```

© 2016 IBM Corporation

Branching

- Super fast branching
- Git's “killer feature”
- Just a file with a SHA1 reference to a commit (41 bytes of data)
- No copy of files is made
- Commit knows its parents, so merging is easy
- Creates completely new workflows

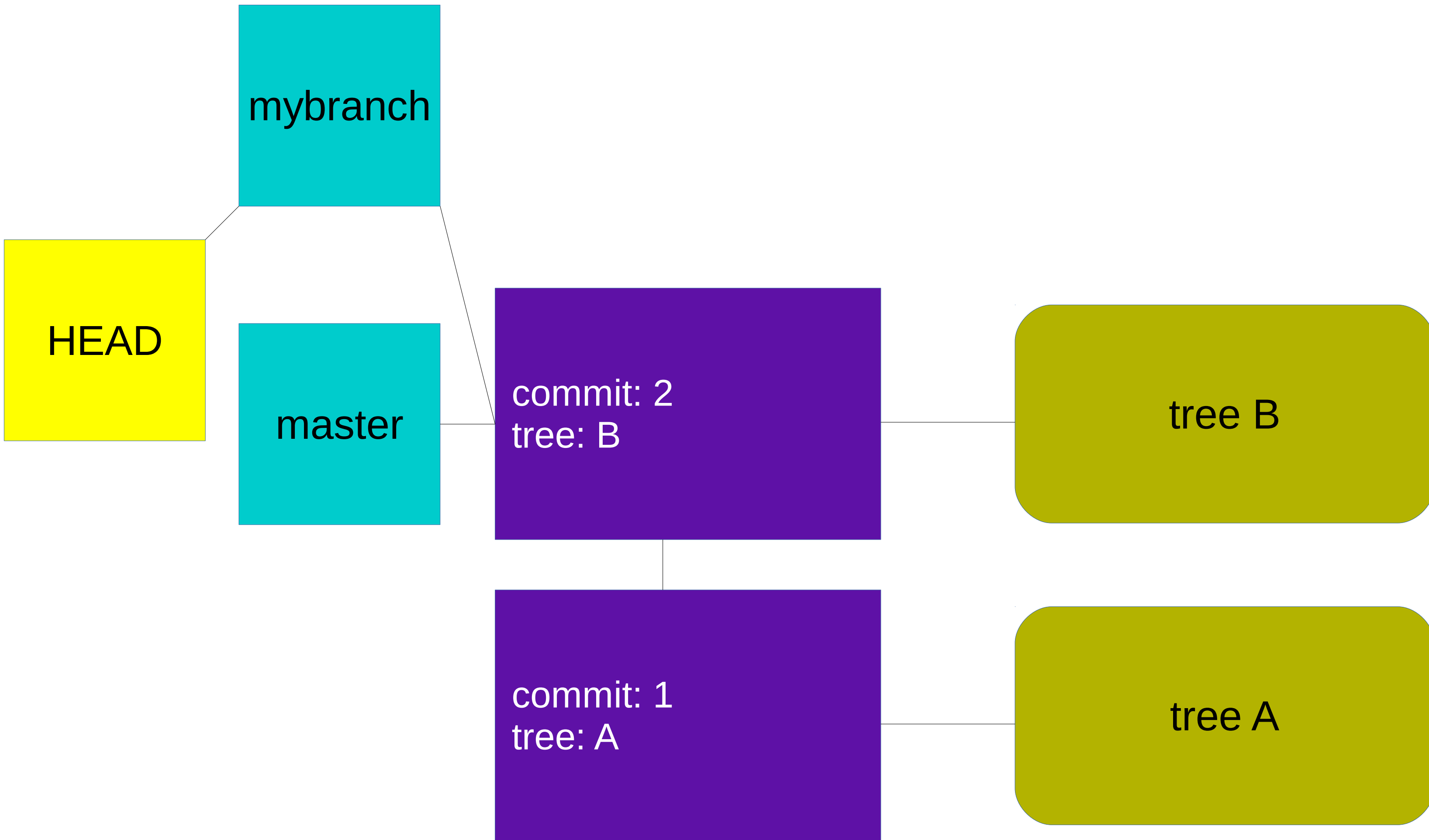
© 2016 IBM Corporation

Creating a branch

```
$ git branch newbranch

$ git checkout newbranch

Switched to branch 'newbranch'

# shortcut: git checkout -b newbranch
```

© 2016 IBM Corporation

Behind the scenes



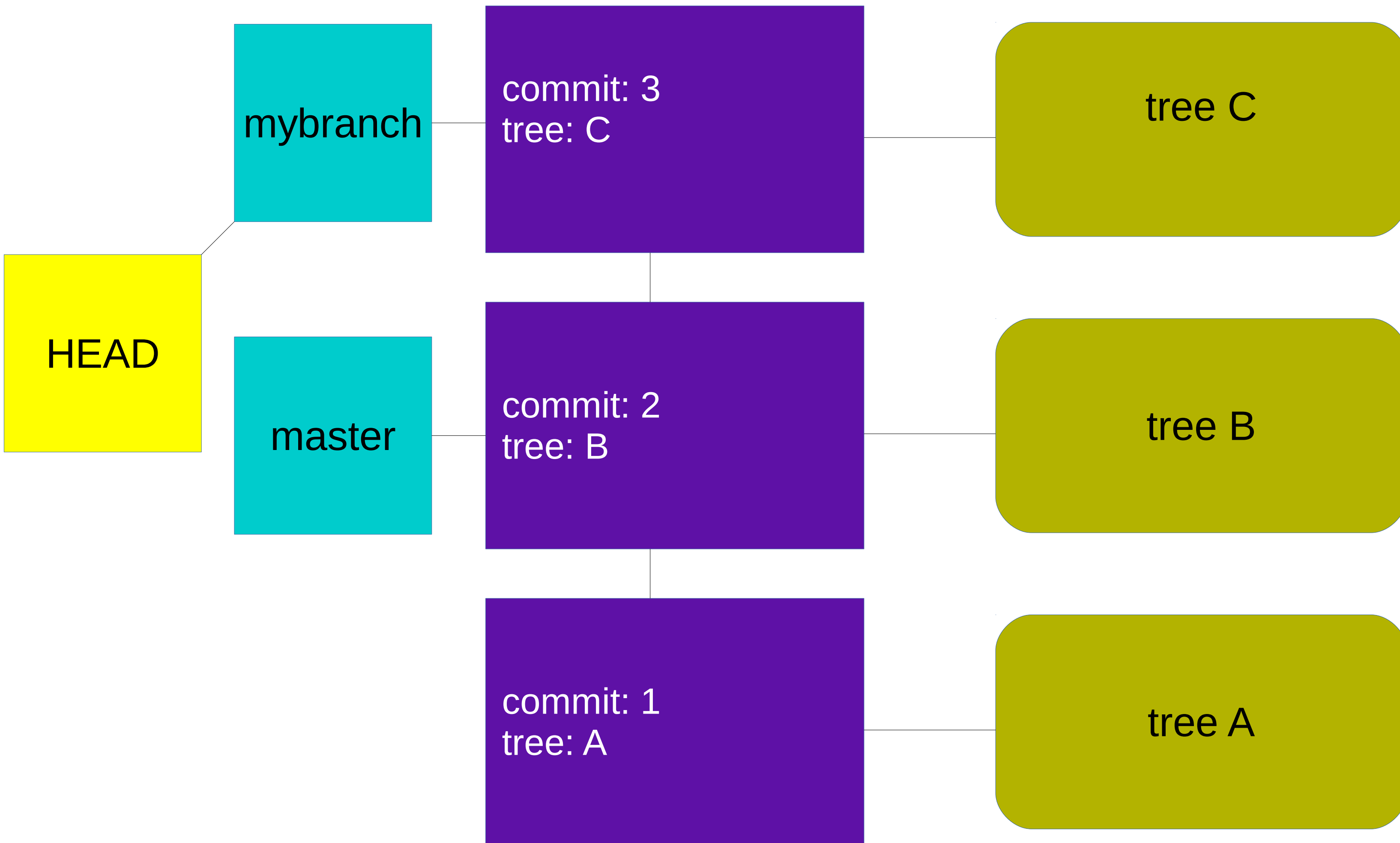
© 2016 IBM Corporation

Making changes in a branch

```
$ echo "My name is Foo" > foo

$ git commit -a -m 'Add some test data to foo'
```

© 2016 IBM Corporation

Behind the scenes



© 2016 IBM Corporation

Creating a branch

```
$ git log --pretty=oneline --abbrev-commit
38ad93f Add some test data to foo
3d1f069 Don't need baz
c947b45 Add some test data to bar
e287cdc Add foo, bar, and baz

$ git checkout master
$ git log --pretty=oneline --abbrev-commit
3d1f069 Don't need baz
c947b45 Add some test data to bar
e287cdc Add foo, bar, and baz
```

© 2016 IBM Corporation

Merging it back

```
$ git checkout master
$ git merge newbranch
Updating 3d1f069..38ad93f
Fast-forward
 foo | 1 +
 1 file changed, 1 insertion(+)
```

© 2016 IBM Corporation

Merging

- Multiple merging strategies
  - *fast-forward
  - recursive
  - resolve
  - octopus (octomerge)

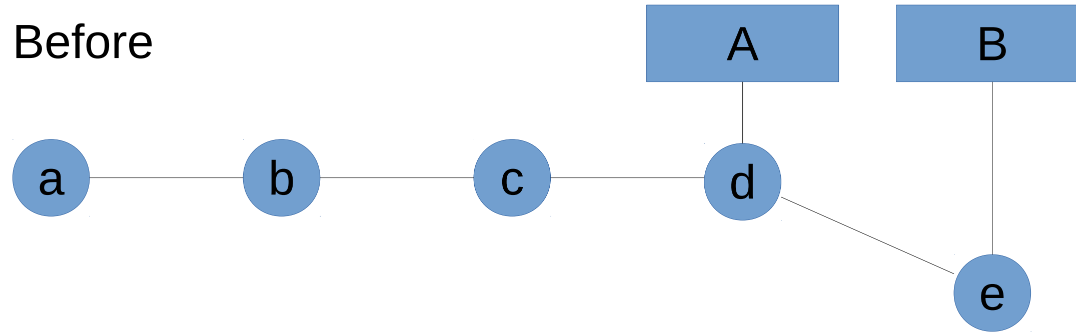
© 2016 IBM Corporation

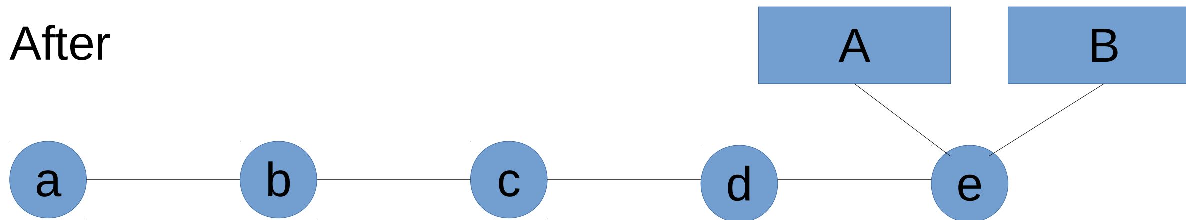
Fast Forward “Merging”

- Branch A is a complete subset of branch B
- Branch B is being merged in to A
- Branch A is said to be “behind” B
- Branch label is moved forward to the label of branch B
- No new “merge” commit created

© 2016 IBM Corporation

Fast Forward “Merging”





© 2016 IBM Corporation

Diverging a bit

```
$ git checkout master
$ sed -i 's|Foo|foo|g' foo
$ git commit -a -m 'Fix typo in foo'
$ echo "'$(cat foo)'      '$(cat bar)'"
'My name is foo'      'My name is Bar'


$ git checkout newbranch
$ sed -i 's|Bar|bar|g' bar
$ git commit -a -m 'Fix typo in bar'
$ echo "'$(cat foo)'      '$(cat bar)'"
'My name is Foo'      'My name is bar'
```

© 2016 IBM Corporation

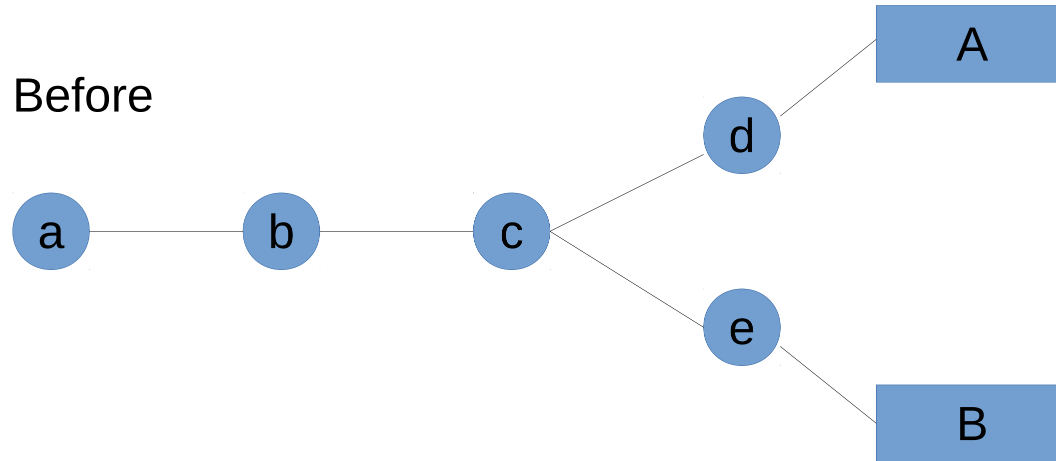
Merging it back

```
$ git checkout master
$ git merge -m "Merge branch 'newbranch'" newbranch
Merge made by the 'recursive' strategy.
 bar | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
$ echo "'$(cat foo)'     '$(cat bar)'"
'My name is foo'     'My name is bar'
$ git log --pretty=oneline --abbrev-commit
372f84a Merge branch 'newbranch'
d7cf47c Fix typo in bar
1dbf752 Fix typo in foo
```

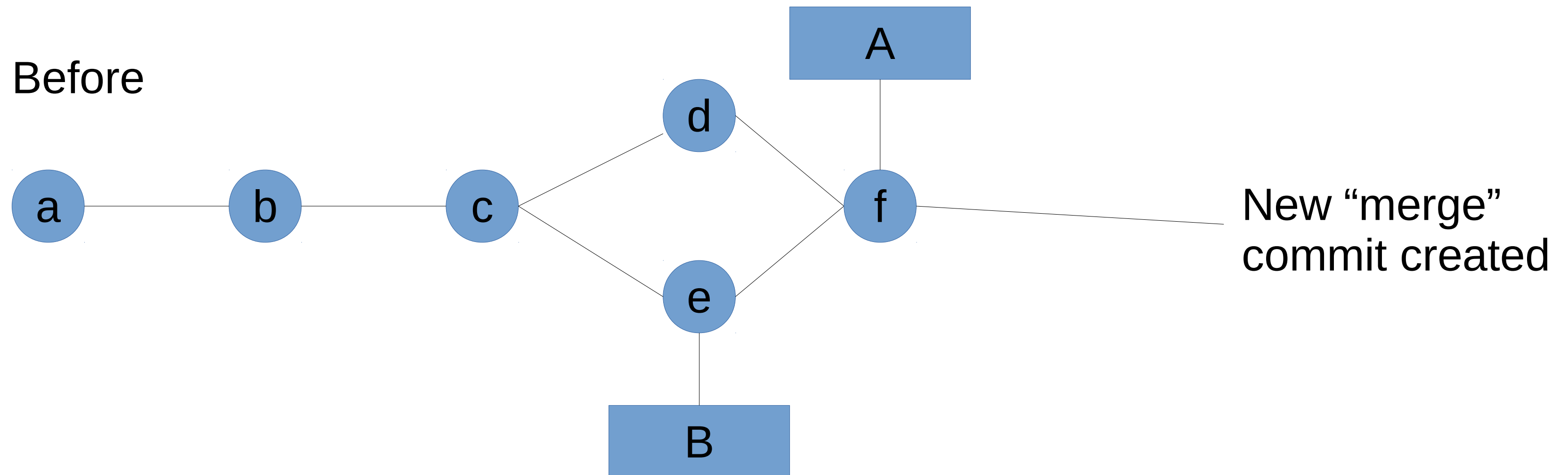
© 2016 IBM Corporation

Recursive Merging

Before



Before



© 2016 IBM Corporation

Diverging even further

```
$ git checkout master
$ echo "No, my name is Kevin" > foo
$ git commit -a -m 'Using my real name'

$ git checkout newbranch
$ echo "No, my name is Kevin Adler" > foo
$ git commit -a -m 'Using my full name'
```

© 2016 IBM Corporation

Diverging even further

```
$ git checkout master
Switched to branch 'master'
$ git merge -m "Merge branch 'newbranch'" newbranch
Auto-merging foo
CONFLICT (content): Merge conflict in foo
Automatic merge failed; fix conflicts and then commit the result.
```

© 2016 IBM Corporation

Conflict resolution

- Sometimes conflicts happen when merging
- git has many merging strategies to cope with conflicts
- Sometimes, that's not enough :(
  - Need arbitration
  - Who's the arbiter? You!

© 2016 IBM Corporation

Resolving conflicts

1) Find the conflicts
   - git status shows unmerged paths
   - Files will contain the lines which have conflicts
   - Conflicts are marked by markers

```
<<<<<<< HEAD
# changes from this branch
=======
# changes from branch being merged
>>>>>>> branchname
```

© 2016 IBM Corporation

Resolving conflicts

2) Once you've found the conflicts, you need to resolve them
   - pick the current branch's code
   - pick the merging branch's code
   - merge the code in to something new
3) Remove the conflict markers
4) Mark the file as resolved
5) Repeat previous steps until all files are resolved
6) Commit the changes

© 2016 IBM Corporation

Resolving conflicts

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:      foo
```

© 2016 IBM Corporation

Resolving conflicts

```
$ cat foo
<<<<<<< HEAD
No, my name is Kevin
=======
No, my name is Kevin Adler
>>>>>>> newbranch
$ echo 'No, my name is Kevin Adler' > foo
$ git add foo
$ git commit
[master af57378] Merge branch 'newbranch'
```

© 2016 IBM Corporation

Visualizing your log

```
$ git log --graph --decorate --pretty=oneline --abbrev-commit
*   af57378 (HEAD, master) Merge branch 'newbranch'
|\
| * 8e7299a (newbranch) Using my full name
* | cad4c46 Using my real name
|/
*   372f84a Merge branch 'newbranch'
|\
| * d7cf47c Fix typo in bar
* | 1dbf752 Fix typo in foo
|/
* 38ad93f Add some test data to foo
```

© 2016 IBM Corporation

How will I remember that?

https://xkcd.com/1597/

© 2016 IBM Corporation

Aliases, man! Aliases

```
$ git config --global alias.tree "log --graph --decorate
--pretty=oneline --abbrev-commit"

$ git tree

*   af57378 (HEAD, master) Merge branch 'newbranch'
|\
| * 8e7299a (newbranch) Using my full name
* | cad4c46 Using my real name
|/
*   372f84a Merge branch 'newbranch'
|\
| * d7cf47c Fix typo in bar
* | 1dbf752 Fix typo in foo
|/
* 38ad93f Add some test data to foo
```

© 2016 IBM Corporation

Remote repositories

- Each git directory is a repository (local repo)
- Git allows remote repositories
  - git clone automatically creates one (origin)
  - you can have as many as you like
  - multiple protocols supported
    - git
    - ssh
    - http(s)
    - ftp
    - file
- Remotes enable sharing and collaboration
  - push changes to them (git push)
  - pull chnages from them (git pull)

© 2016 IBM Corporation

Bare repositories

- Bare repositories are typically used for remotes
  - contain no working directory or checked out code
  - just git database
  - git database stored in the directory instead of under .git
- Typically use .git extension on the directory

```
$ git init --bare ~/my_project.git

Initialized empty Git repository in
/home/kadler/my_project.git/
```

© 2016 IBM Corporation

Adding a remote

```
$ git remote -v

# no remotes, let's add one

$ git remote add origin /home/kadler/my_project.git

$ git remote -v

origin  /home/kadler/my_project.git/ (fetch)

origin  /home/kadler/my_project.git/ (push)
```

© 2016 IBM Corporation

Sharing is caring

```
$ git push origin master
Counting objects: 26, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (19/19), done.
Writing objects: 100% (26/26), 2.29 KiB | 0 bytes/s, done.
Total 26 (delta 2), reused 0 (delta 0)
To /home/kadler/my_project.git/
 * [new branch]      master -> master
```

© 2016 IBM Corporation

Sharing is caring

```
$ git clone ~/my_project.git ~/my_project2
$ cd ~/my_project2
$ git remote -v
origin  /home/kadler/my_project.git (fetch)
origin  /home/kadler/my_project.git (push)
$ echo '1. add things to todo list' > todo.txt
$ git add todo.txt
$ git commit -a -m 'Add a todo'
$ git push origin master
```

© 2016 IBM Corporation

Sharing is caring

```
$ cd ~/my_project
$ git pull origin master
From /home/kadler/my_project
 * branch            master     -> FETCH_HEAD
Updating af57378..080d547
Fast-forward
 todo.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 todo.txt
```

© 2016 IBM Corporation

Additional Resources

- Git online documentation: https://git-scm.com/doc
- Pro Git online book: https://git-scm.com/book/en/v2
- Setting up SSH keys
  - BitBucket: https://confluence.atlassian.com/bitbucket/set-up-ssh-for-git-728138079.html
  - GitHub: https://help.github.com/articles/generating-an-ssh-key/
- gitolite (ssh-based, local git hosting): http://gitolite.com
- Git for Ages 4 and Up: https://youtu.be/1ffBJ4sVUb4
- Git GUIs:
  - ungit (Node.js Git web GUI): https://github.com/FredrikNoren/ungit
  - GitHub Desktop (Mac and Windows): https://desktop.github.com/
  - SourceTree (Mac and Windows): https://www.sourcetreeapp.com/
  - More: https://git-scm.com/downloads/guis

© 2016 IBM Corporation

Questions?

© 2015 IBM Corporation

Advanced Topics

© 2015 IBM Corporation

Adding Tags

- Two types, lightweight and annotated
- Lightweight
  - Basically a “named commit”
  - Like a branch, but never moves
- Annotated
  - Full object
    - tagger name
    - tagger email
    - tag date
    - tag message
  - Checksummed
  - Can be signed and verified using GNU Privacy Guard

© 2016 IBM Corporation

Stashing

- A stack of temporary branches
- Useful if you're in the middle of changes and need to work on something else
- Don't want to commit the changes yet? Just stash them

```
$ git stash
Saved working directory and index state WIP on master:
c87907d Add changelog
HEAD is now at c87907d Add changelog
$ git stash list
stash@{0}: WIP on master: c87907d Add changelog
$ git stash pop
```

© 2016 IBM Corporation

Rewriting (local) history

- Want to change your history? git reset!
- Conceptually simple: moves HEAD to the given commit
- What happens to your working directory?
  - --soft: working tree and staging area left intact
  - --mixed: working tree left intact, staging area cleared (default)
  - --hard: working tree and staging area cleared (**CAUTION**)

```
$ git reset HEAD^ # reset to the previous commit
```

© 2016 IBM Corporation

Making (local) Amends

- You can amend a commit, provided you haven't pushed it already.
- Works very well with git reset --soft

```
# make changes to files
# git add, ...

$ git commit --amend
```

© 2016 IBM Corporation

Incremental changes

- Don't want to add all the changes you've made?
- git add --patch (-p)
- Requires Perl support

```
$ git diff
diff --git a/changelog b/changelog
index a4033a2..3fefa05 100644
--- a/changelog
+++ b/changelog
@@ -1 +1,2 @@
-2016-01-01 - created stfuf
+2016-01-05 - fixed more stuff
+2016-01-01 - created stuff
```

© 2016 IBM Corporation

Incremental changes

```
$ git add -p
diff --git a/changelog b/changelog
index a4033a2..3fefa05 100644
--- a/changelog
+++ b/changelog
@@ -1 +1,2 @@
-2016-01-01 - created stfuf
+2016-01-05 - fixed more stuff
+2016-01-01 - created stuff
Stage this hunk [y,n,q,a,d,/,e,?]? e
# edit the hunk and remove the fixed more stuff
```

© 2016 IBM Corporation

Incremental changes

```
$ git diff --cached
--- a/changelog
+++ b/changelog
@@ -1 +1 @@
-2016-01-01 - created stfuf
+2016-01-01 - created stuff
$ git diff
--- a/changelog
+++ b/changelog
@@ -1 +1,2 @@
+2016-01-05 - fixed more stuff
 2016-01-01 - created stuff
```

© 2016 IBM Corporation

Incremental changes

```
Stage this hunk [y,n,q,a,d,/,e,?]? ?
y - stage this hunk
n - do not stage this hunk
q - quit; do not stage this hunk nor any of the remaining ones
a - stage this hunk and all later hunks in the file
d - do not stage this hunk nor any of the later hunks in the file
g - select a hunk to go to
/ - search for a hunk matching the given regex
j - leave this hunk undecided, see next undecided hunk
J - leave this hunk undecided, see next hunk
k - leave this hunk undecided, see previous undecided hunk
K - leave this hunk undecided, see previous hunk
s - split the current hunk into smaller hunks
e - manually edit the current hunk
? - print help
```

© 2016 IBM Corporation

Cherry-pick your battles

- Instead of merging an entire branch, need just one commit
- Useful for pulling bugfixes from service branches to master or vice-versa

```
$ git cherry-pick 080d547
[newbranch adf05fc] Add a todo
 1 file changed, 1 insertion(+)
 create mode 100644 todo.txt
```

© 2016 IBM Corporation

Billy's Git trail

- Git keeps track of where you've been
- Amend a commit and want to get back to the original? Find it in the reflog.

```
git reflog

c87907d HEAD@{0}: commit: Add changelog

080d547 HEAD@{1}: reset: moving to
080d54779f09700cd93e151c13cdd1c4f4cbb8f4

964ce19 HEAD@{2}: commit: create changelog

91405f4 HEAD@{3}: commit: Add changelog

080d547 HEAD@{4}: reset: moving to 080d547


git reflog –all # show all references, even orphans
```

© 2016 IBM Corporation