`https://xkcd.com/1597/` on Git

If that doesn't fix it, git.txt contains the phone number of a friend of mine who understands git. Just wait through a few minutes of "It's really pretty simple, just think of branches as..." and eventually you'll learn the commands that will fix everything.

Git != GitHub

Today is about Git, not GitHub

Part 1: Working Tree, Repository, and Staging Area

Start Work

```
$ mkdir r1 && cd r1
$ echo hi > greeting.txt
$ echo bye > parting.txt
$ tree -a
.
├── greeting.txt
└── parting.txt
```

Start Work

```
$ mkdir r1 && cd r1
$ echo hi > greeting.txt
$ echo bye > parting.txt
$ tree -a
.
├── greeting.txt
└── parting.txt
```

Like `ls -a`,
but prettier and
with subdirectories

Git Repository

A Git repository is "just" a directory

```
$ git init && git checkout -b main
Initialized empty Git repository ...
Switched to a new branch 'main'
$ tree -a
.
├── .git
│   ├── ...
│   └── ...
├── greeting.txt
└── parting.txt
```

Git Repository

A Git repository is "just" a directory

```
$ git init && git checkout -b main
Initialized empty Git repository ...
Switched to a new branch 'main'
$ tree -a
.
├── .git
│   ├── ...
│   └── ...
├── greeting.txt
└── parting.txt
```

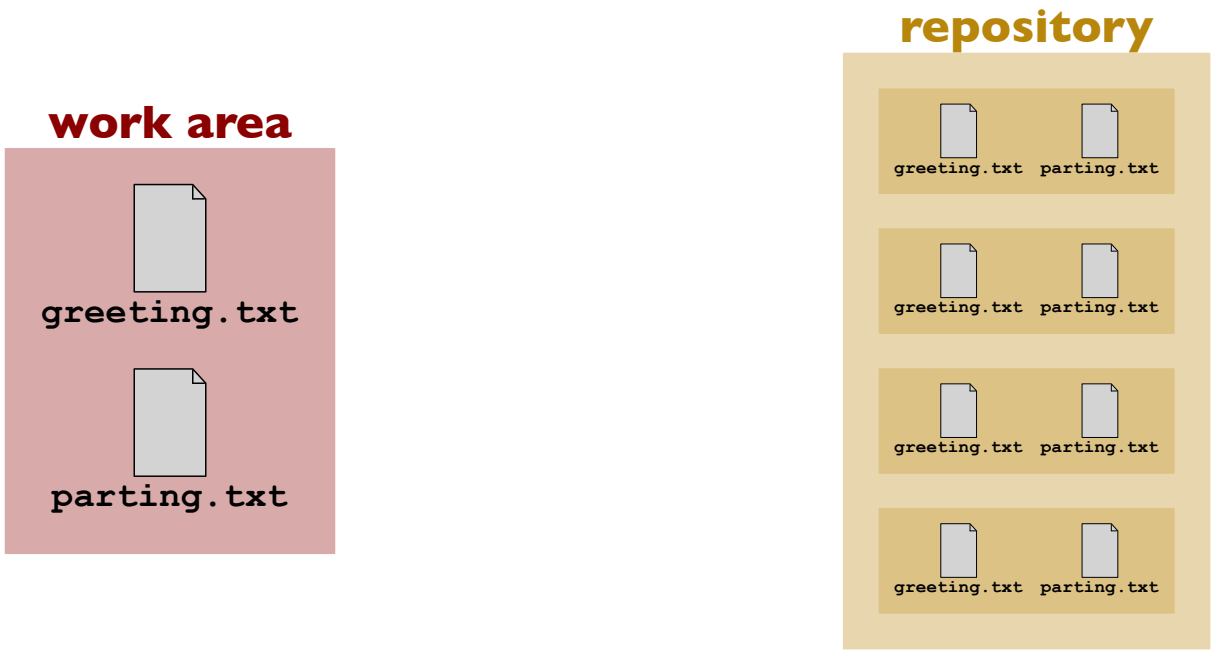
repository

Git Repository

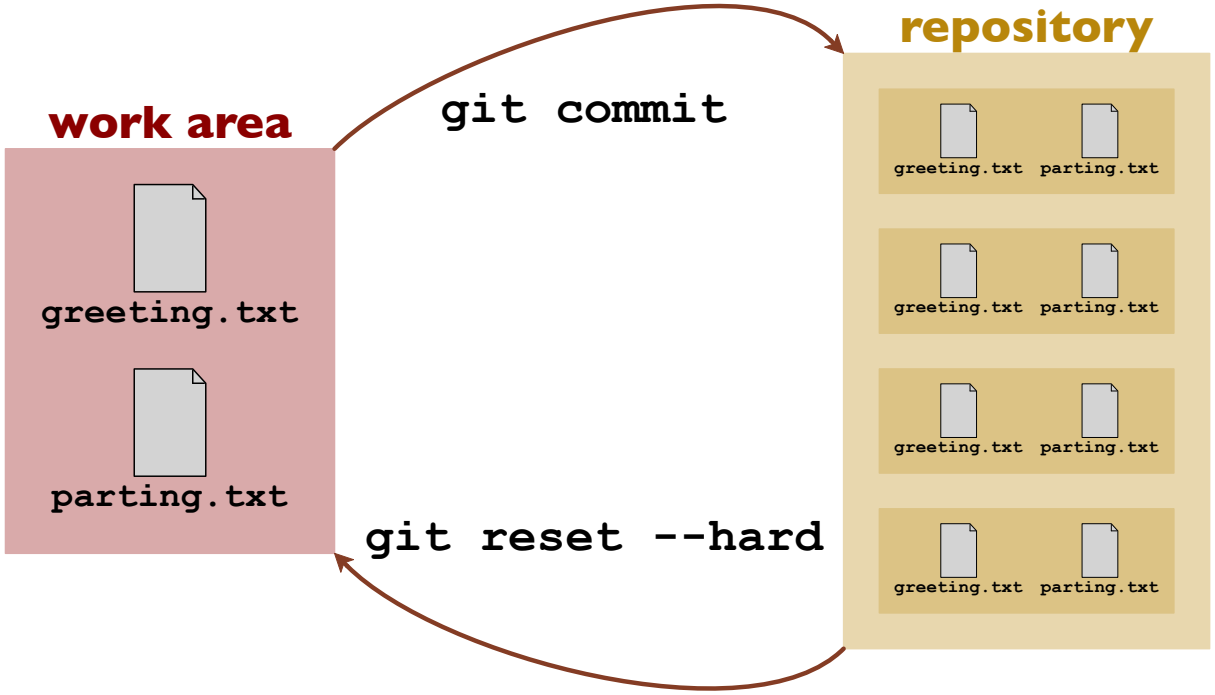
A Git repository is "just" a directory

```
$ git init && git checkout -b main
Initialized empty Git repository ...
Switched to a new branch 'main'
$ tree -a
.
├── .git
│   ├── ...
│   └── ...
├── greeting.txt
└── parting.txt
```

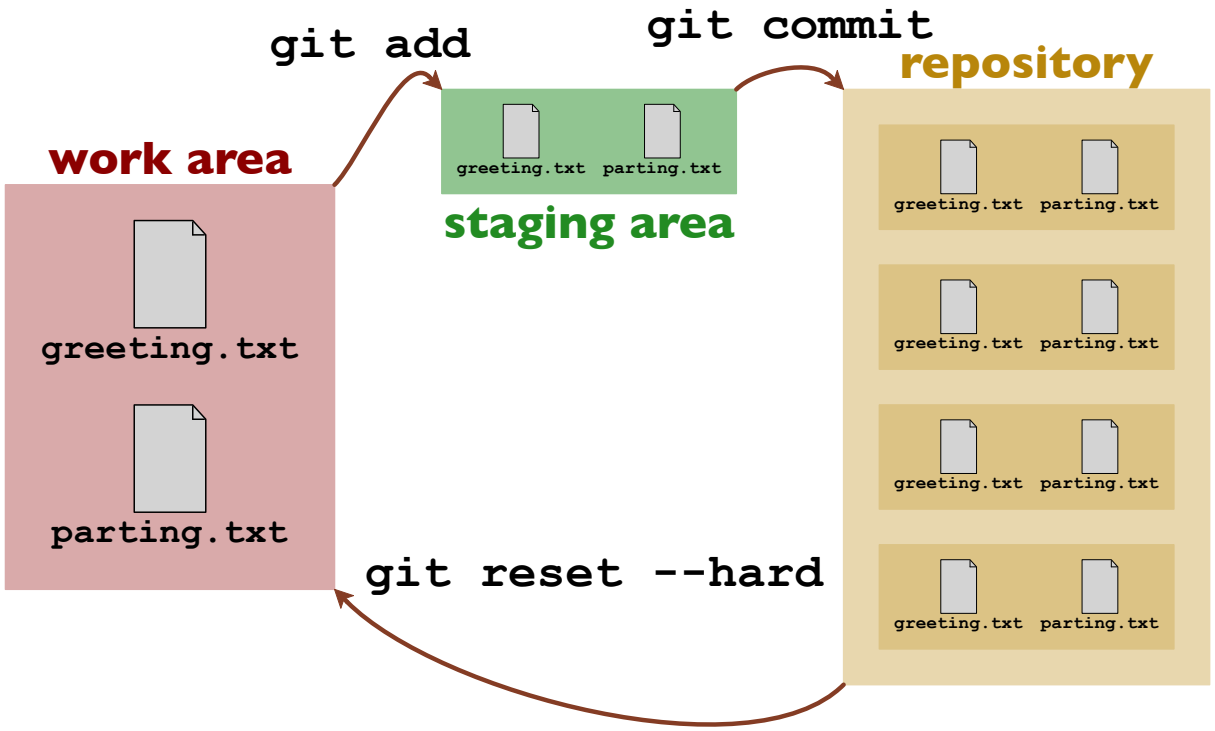
Git Workflow



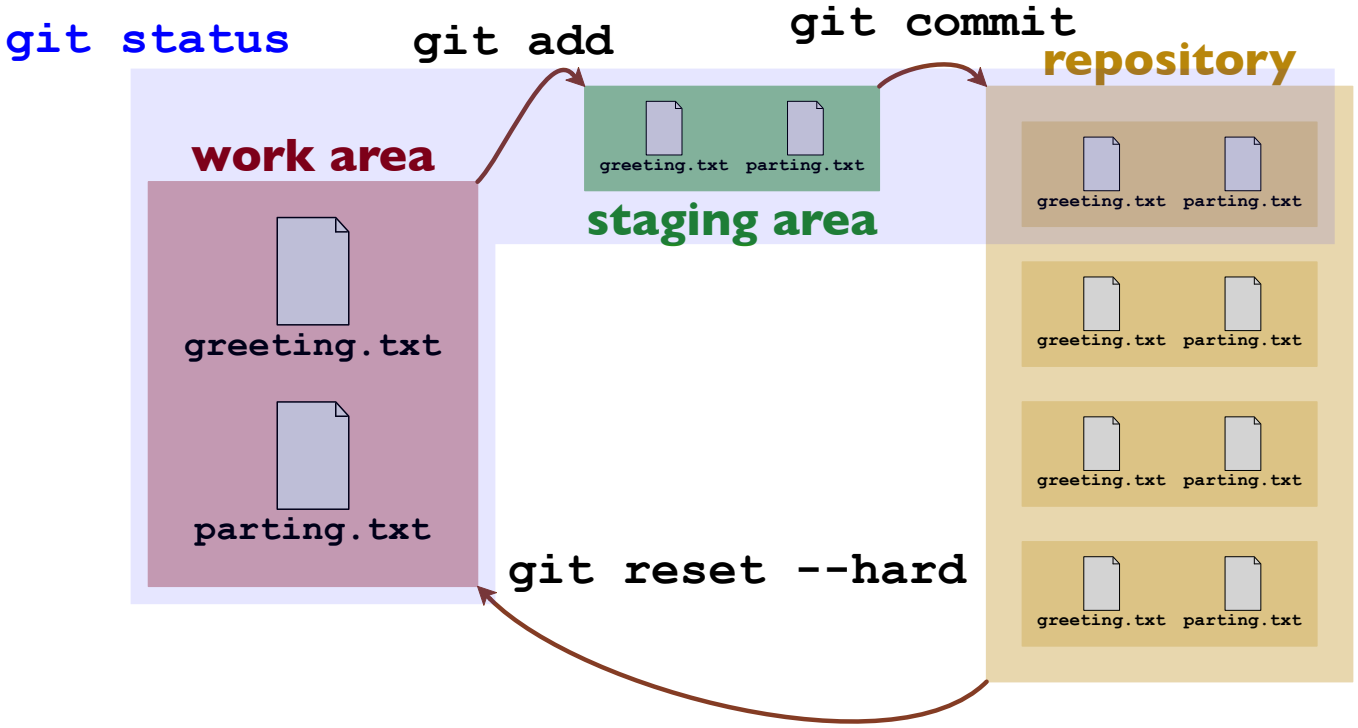
Git Workflow



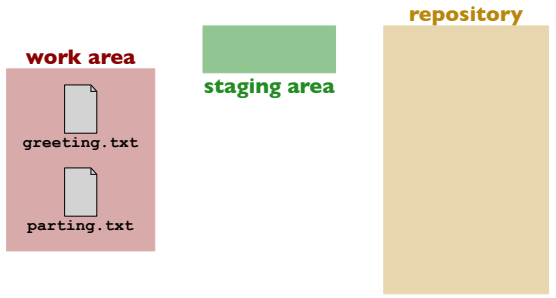
Git Workflow



Git Workflow

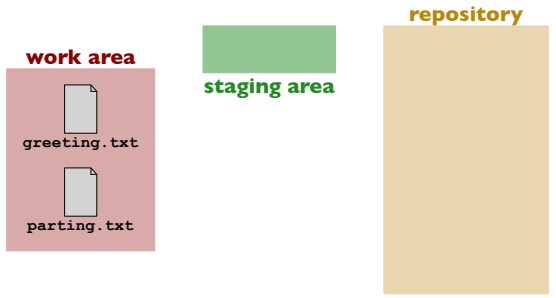


Checking the Status




```
$ git status
On branch main

No commits yet

Untracked files:
  greeting.txt
  parting.txt
```

Checking the Status



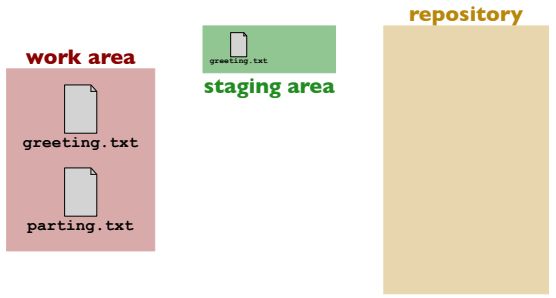

```
$ git status
On branch main

No commits yet

Untracked files:
  greeting.txt
  parting.txt
```

commit in repository

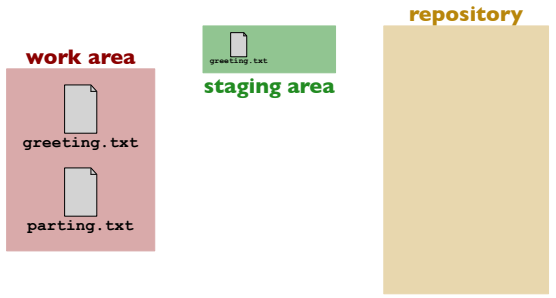
workarea difference from **staging area**

Checking the Status

```
$ git add greeting.txt

$ git status
On branch main

No commits yet

Changes to be committed:
  new file:   greeting.txt

Untracked files:
  parting.txt
```



Checking the Status



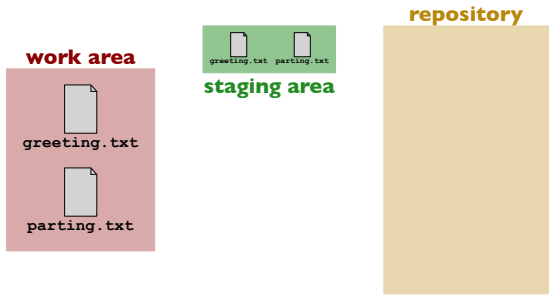

```
$ git add greeting.txt

$ git status
On branch main
No commits yet

Changes to be committed:
  new file:   greeting.txt

Untracked files:
  parting.txt
```

commit in repository

staging area difference from **commit**

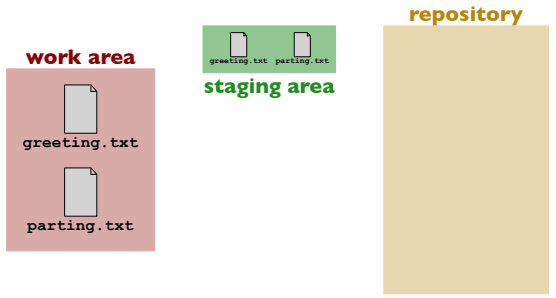
workarea difference from **staging area**

Checking the Status




```
$ git add parting.txt

$ git status
On branch main

No commits yet

Changes to be committed:
  new file:   greeting.txt
  new file:   parting.txt
```

Checking the Status

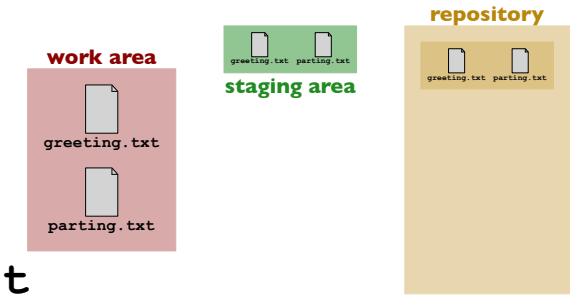



```
$ git add parting.txt

$ git status
On branch main
No commits yet

Changes to be committed:
  new file:   greeting.txt
  new file:   parting.txt
```

commit in repository

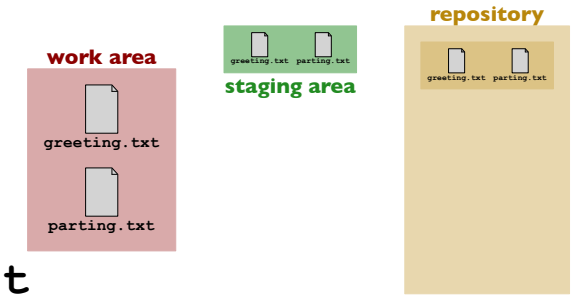
staging area difference from **commit**

Checking the Status




```
$ git commit -m first
[main (root-commit) 04be3f4] first
2 files changed, 2 insertions(+)
create mode 100644 greeting.txt
create mode 100644 parting.txt

$ git status
On branch main
nothing to commit, working tree clean
```

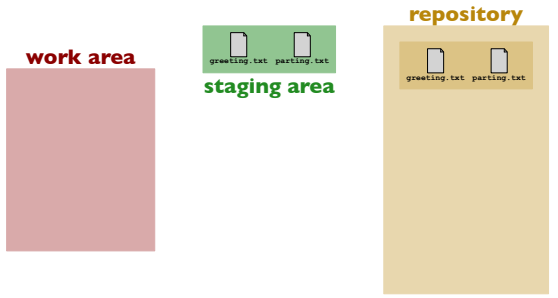
Checking the Status




```
$ git commit -m first
[main (root-commit) 04be3f4] first
2 files changed, 2 insertions(+)
create mode 100644 greeting.txt
create mode 100644 parting.txt

$ git status
On branch main
nothing to commit, working tree clean
```

commit in repository

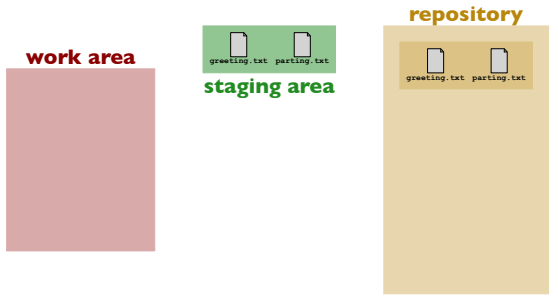
Checking the Status



A Git repository is "just" a directory

```
$ mkdir ../r2 && cd ../r2
$ cp -r ../r1/.git .git

$ git status
On branch main
Changes not staged for commit:
 deleted:    greeting.txt
 deleted:    parting.txt
```

Checking the Status



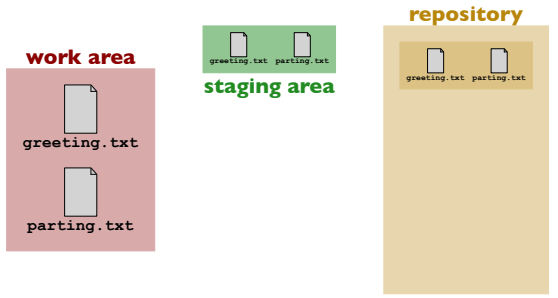
A Git repository is "just" a directory

```
$ mkdir ../r2 && cd ../r2
$ cp -r ../r1/.git .git

$ git status
On branch main
Changes not staged for commit:
  deleted:    greeting.txt
  deleted:    parting.txt
```

commit in repository

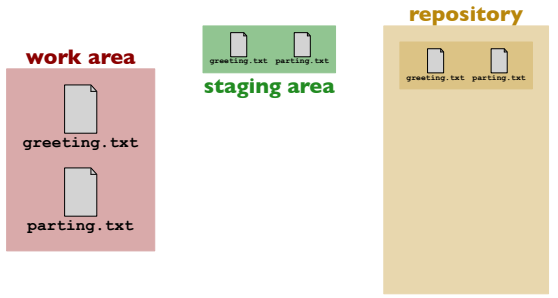
workarea difference from **staging area**

Checking the Status




```
$ git reset --hard
HEAD is now at 04be3f4 first

$ git status
On branch main
nothing to commit, working tree clean
```

Checking the Status




```
$ git reset --hard
HEAD is now at 04be3f4 first

$ git status
On branch main
nothing to commit, working tree clean
```

commit in repository

Cloning

Using **`git clone`** is usually better than **`cp -r`**:

```
$ cd ..
$ git clone r1 r3
Cloning into 'r3'...
done.
$ cd r3
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

Cloning

Using **`git clone`** is usually better than **`cp -r`**:

```
$ cd ..
$ git clone r1 r3
Cloning into 'r3'...
done.
$ cd r3
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

Almost the same as
`cp -r r2/.git r3/.git`
plus **`git reset --hard`** …

Cloning

Using **`git clone`** is usually better than **`cp -r`**:

```
$ cd ..
$ git clone r1 r3
Cloning into 'r3'...
done.
$ cd r3
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

... but remembers **`r2`** as **`origin`**
for future **`git pull`** and **`git push`**

Cloning

```
$ git clone https://github.com/UtahMSD/r1
```

vs.

```
$ git clone r1
```

is similar to viewing

https://msd.utah.edu/index.html

vs.

index.html

Part 2: Git Internals

How Git Works

Git's implementation uses these key ideas:

`diff`

compare two files and extract the difference

`patch`

apply a diff to a file that may have other changes already

Object hashing with SHA1

reliably summarize content with a short identifier

Git Repository Files

```
$ git init && git checkout -b main
```

Git Repository Files

```
$ git init && git checkout -b main
$ tree .git
.git
├── HEAD
├── config
├── description
├── hooks
│   └── ....
├── info
│   └── exclude
├── objects
│   ├── info
│   └── pack
└── refs
    ├── heads
    └── tags
```

Git Repository Files

```
$ git init && git checkout -b main
$ tree .git
.git
├── HEAD
├── config
├── description
├── hooks
│   └── ....
├── info
│   └── exclude
├── objects
│   ├── info
│   └── pack
└── refs
    ├── heads
    └── tags
```

Git Repository Files

```
$ git init && git checkout -b main
$ tree .git
.git
├── HEAD
├── config
├── description
├── hooks
│   └── ....
├── info
│   └── exclude
├── objects
│   ├── info
│   └── pack
└── refs
    ├── heads
    └── tags
```

key–value store

Keys for Values

greeting.txt

```
Hello, World!
```

```
$ git hash-object greeting.txt
8ab686eafeb1f44702738c8b0f24f2567c36da6d
```

```
$ printf 'blob 14\0Hello, World!\n' | openssl sha1
8ab686eafeb1f44702738c8b0f24f2567c36da6d
```

Storing Objects

```
$ git hash-object -w greeting.txt
8ab686eafeb1f44702738c8b0f24f2567c36da6d

$ tree .git
.git
├── ....
├── objects
│   ├── 8a
│   │   └── b686eafeb1f44702738c8b0f24f2567c36da6d
│   ├── info
│   └── pack
└── ....
```

Showing Objects

```
$ git show 8ab686eafeb1f44702738c8b0f24f2567c36da6d
Hello, World!

$ git show 8ab68
Hello, World!

$ git cat-file -p 8ab68
Hello, World!
```

Different Content ⇒ Different Object

```
$ cat > greeting.txt
Hi, everybody!

$ git hash-object -w greeting.txt
362cd1d3448db147f56ee294ccfca422f136a758

$ tree .git
.git
├── ....
├── objects
│   ├── 36
│   │   └── 2cd1d3448db147f56ee294ccfca422f136a758
│   ├── 8a
│   │   └── b686eafeb1f44702738c8b0f24f2567c36da6d
│   ├── info
│   └── pack
└── ....
```

Cleanup...

```
$ rm -rf .git/objects/36 .git/objects/8a
$ tree .git
.git
├── ....
├── objects
│   ├── info
│   └── pack
└── ....
```

... but never do that again!

Adding a File Creates an Object

```
$ git add greeting.txt
$ tree .git
.git
├── ....
├── index
├── ....
├── objects
│   ├── 36
│   │   └── 2cd1d3448db147f56ee294ccfca422f136a758
│   ├── info
│   └── pack
└── ....
```

Adding a File Creates an Object

```
$ git add greeting.txt
$ tree .git
.git
├── ....
├── index
├── ....
├── objects
│   ├── 36
│   │   └── 2cd1d3448db147f56ee294ccfca422f136a758
│   ├── info
│   └── pack
└── ....
```

Adding a File Creates an Object

```
$ git add greeting.txt
$ tree .git
.git
├── ....
├── index
├── ....
├── objects
│   ├── 36
│   │   └── 2cd1d3448db147f56ee294ccfca422f136a758
│   ├── info
│   └── pack
└── ....
```

staging area a.k.a. "index"

Commits Create More Objects

```
$ git commit -m "Dr. Nick's first commit"
[main (root-commit) 05a0cef] Dr. Nick's first commit
....
$ tree .git
.git
├── ....
├── objects
│   ├── 05
│   │   └── a0ceffc8844d40c82b936b7dd56e4d76e22bbf
│   ├── 36
│   │   └── 2cd1d3448db147f56ee294ccfca422f136a758
│   ├── eb
│   │   └── 2769b3d285ebe1eae5d38251b94ba8828da1b8
│   ├── info
│   └── pack
└── ....
```

Commits Create More Objects

```
$ git commit -m "Dr. Nick's first commit"
[main (root-commit) 05a0cef] Dr. Nick's first commit
....
$ tree .git
.git
├── ....
├── objects
│   ├── 05
│   │   └── a0ceffc8844d40c82b936b7dd56e4d76e22bbf
│   ├── 36
│   │   └── 2cd1d3448db147f56ee294ccfca422f136a758
│   ├── eb
│   │   └── 2769b3d285ebe1eae5d38251b94ba8828da1b8
│   ├── info
│   └── pack
└── ....
```

varies depending on your name, email address, and the time

Commits Create More Objects

```
$ git commit -m "Dr. Nick's first commit"
[main (root-commit) 05a0cef] Dr. Nick's first commit
....
$ tree .git
.git
├── ....
├── objects
│   ├── 05
│   │   └── a0ceffc8844d40c82b936b7dd56e4d76e22bbf
│   ├── 36
│   │   └── 2cd1d3448db147f56ee294ccfca422f136a758
│   ├── eb
│   │   └── 2769b3d285ebe1eae5d38251b94ba8828da1b8
│   ├── info
│   └── pack
└── ....
```

commit object

Commits Create More Objects

```
$ git commit -m "Dr. Nick's first commit"
[main (root-commit) 05a0cef] Dr. Nick's first commit
....
$ tree .git
.git
├── ....
├── objects
│   ├── 05
│   │   └── a0ceffc8844d40c82b936b7dd56e4d76e22bbf
│   ├── 36
│   │   └── 2cd1d3448db147f56ee294ccfca422f136a758
│   ├── eb
│   │   └── 2769b3d285ebe1eae5d38251b94ba8828da1b8
│   ├── info
│   └── pack
└── ....
```

tree object

Viewing Different Types of Objects

```
$ git cat-file -p 362cd1
Hi, everybody!
```

A *blob* object is unnamed file content

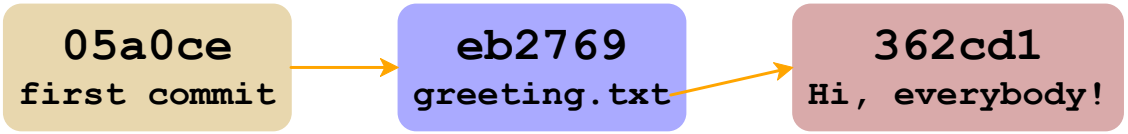
```
$ git cat-file -p eb2769
100644 blob 362cd1d3448db147f56ee294ccfca422f136a758    greeting.txt
```

A *tree* object organizes blobs into a directory-like tree structure

```
$ git cat-file -p 05a0ce
tree eb2769b3d285ebe1eae5d38251b94ba8828da1b8
author Matthew Flatt <mflatt@racket-lang.org> 1582129442 -0700
committer Matthew Flatt <mflatt@racket-lang.org> 1582129442 -0700

Dr. Nick's first commit
```

A *commit* object connects a tree with author and message information

Objects in the Store



Another Commit

```
$ cat > parting.txt
Goodbye.
$ git add parting.txt
$ git commit -m "new file"
[main 73aee29] new file
....
$ tree .git
.git
├── ....
├── objects
│   ├── 05 – a0ceffc8844d40c82b936b7dd56e4d76e22bbf
│   ├── 20 – a440efb4f350a53983ab89485f40c62e913097
│   ├── 36 – 2cd1d3448db147f56ee294ccfca422f136a758
│   ├── 3e – 911a4b3cfb1f2d9d02b56deee316fbfa8e764c
│   ├── 73 – aee2937a7fe941d79c855386e599739fe89b15
│   ├── eb – 2769b3d285ebe1eae5d38251b94ba8828da1b8
│   ├── info
│   └── pack
└── ....
```

Commit History = Pointer to Previous Commit

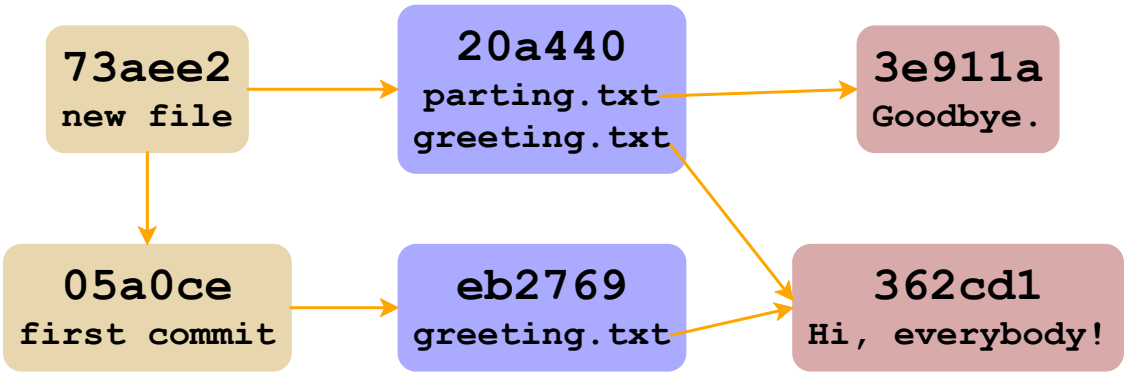
```
$ git cat-file -p 73aee2
tree 20a440efb4f350a53983ab89485f40c62e913097
parent 05a0ceffc8844d40c82b936b7dd56e4d76e22bbf
author Matthew Flatt <mflatt@racket-lang.org> 1582132992 -0700
committer Matthew Flatt <mflatt@racket-lang.org> 1582132992 -0700

new file
```

each commit (except the first) points to a parent commit plus a tree

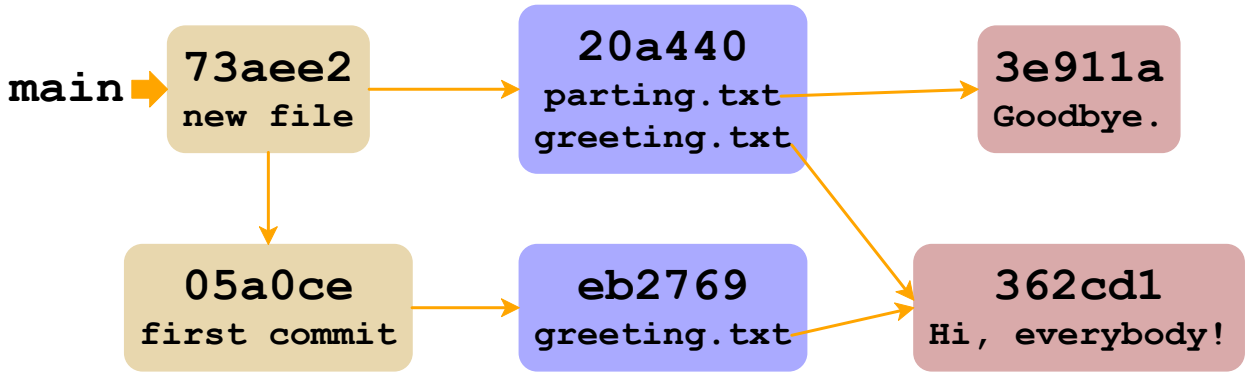
```
$ git cat-file -p 20a440
100644 blob 362cd1d3448db147f56ee294ccfca422f136a758    greeting.txt
100644 blob 3e911a4b3cfb1f2d9d02b56deee316fbfa8e764c    parting.txt
```

a tree points to its content — not a "previous" tree

More Objects in the Store

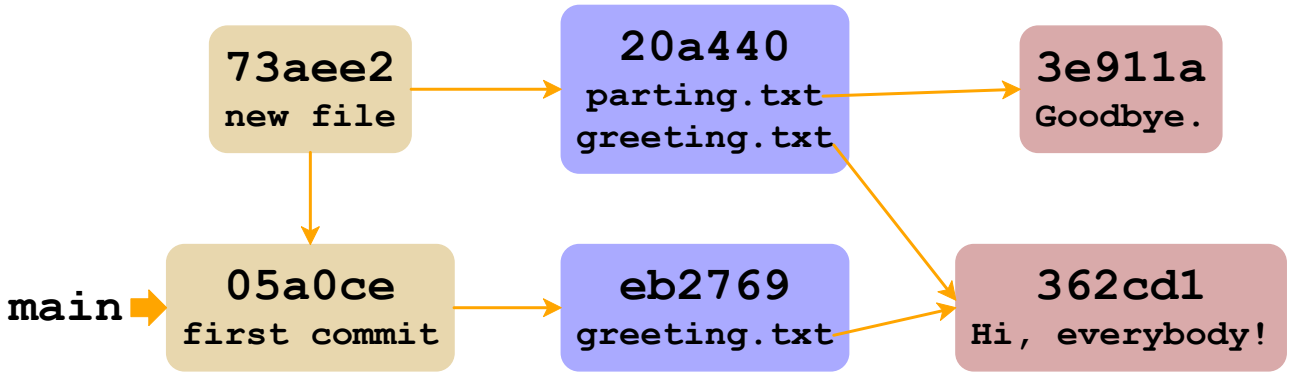


More Objects in the Store



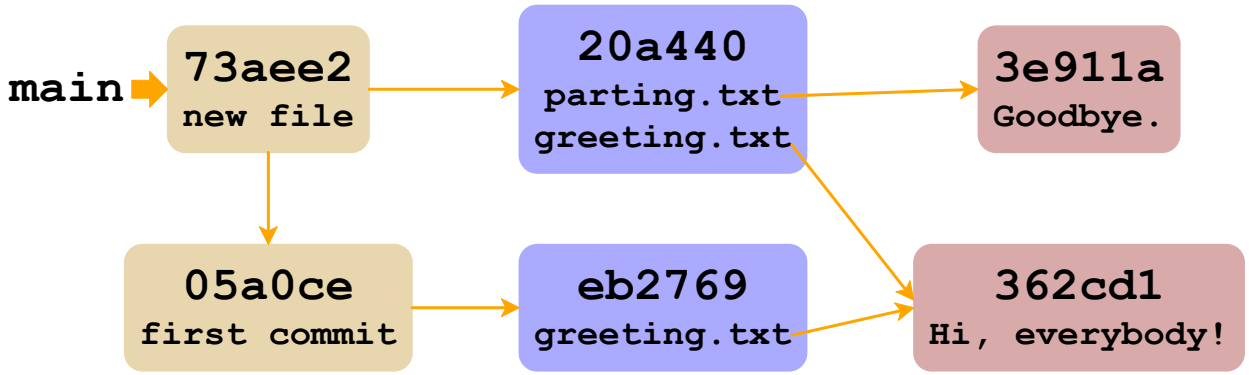

```
$ cat .git/refs/heads/main
73aee2937a7fe941d79c855386e599739fe89b15
```

More Objects in the Store




```
$ git reset 05a0ce

$ cat .git/refs/heads/main
05a0ceffc8844d40c82b936b7dd56e4d76e22bbf
```

More Objects in the Store

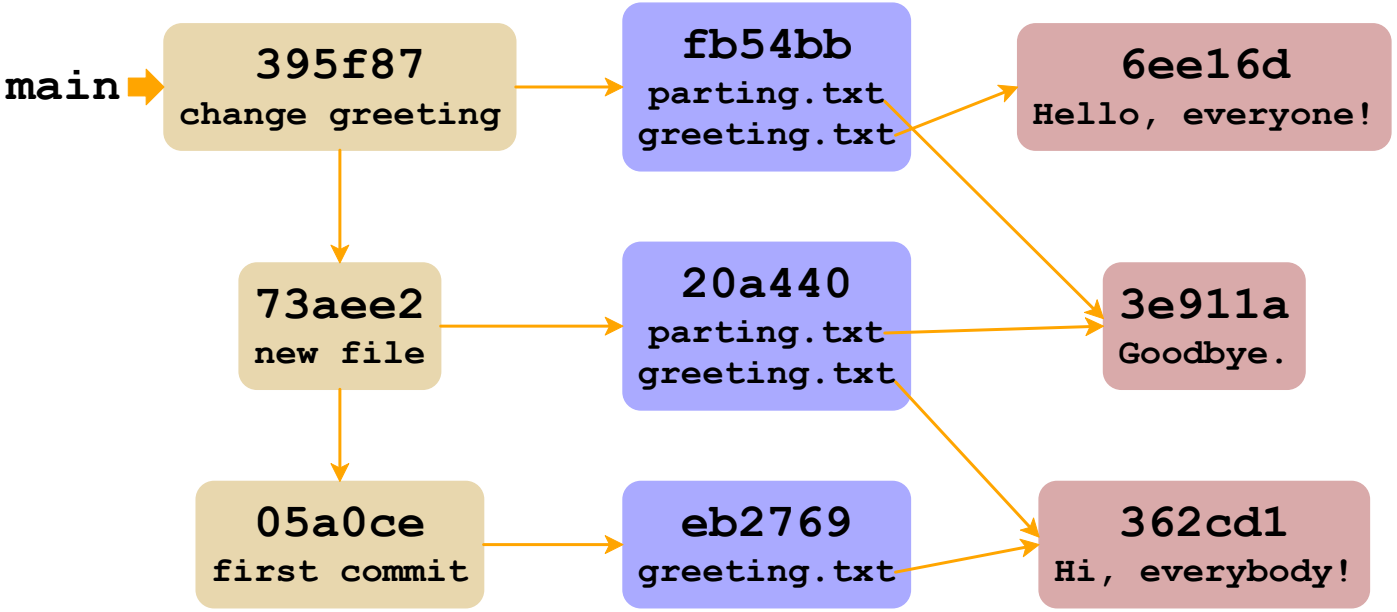



```
$ git reset 73aee2
```

Modifying a File

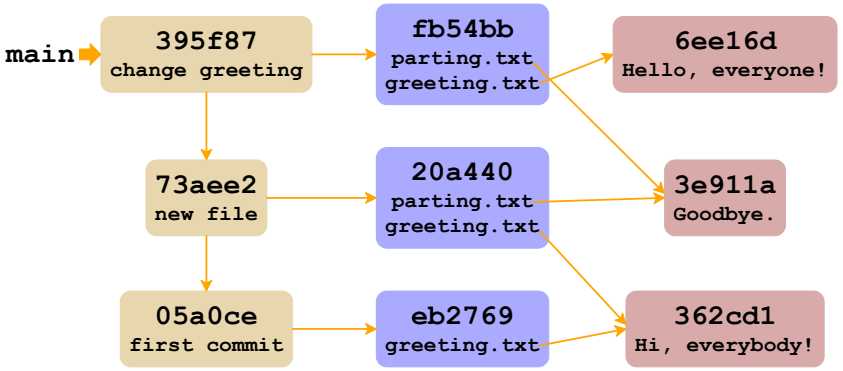
```
$ cat > greeting.txt
Hello, everyone!

$ git hash-object greeting.txt
6ee16d02b2b45c95ee19858d88b615f80b431944

$ git add greeting.txt
$ git commit -m "change greeting"
[main 395f876] change greeting
1 file changed, 1 insertion(+), 1 deletion(-)
```

Updated Object Store



So far, nothing has relied on **`diff`** or **`patch`**...

Diffing Trees



Shows the difference between the current commit and its parent

```
$ git show
...
diff --git a/greeting.txt b/greeting.txt
index 362cd1d..6ee16d0 100644
--- a/greeting.txt
+++ b/greeting.txt
@@ -1 +1 @@
-Hi, everybody!
+Hello, everyone!
```

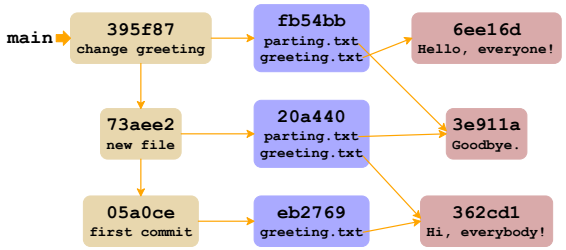
Part 3: Branches

Branches

```
$ git branch intro-prep

$ tree .git
.git
├── ....
└── refs
    ├── heads
    │   ├── intro-prep
    │   └── main
    └── tags

$ cat .git/refs/heads/intro-prep
395f876860c78f435ebf09b94372a35b1bebd01e
```

Branches

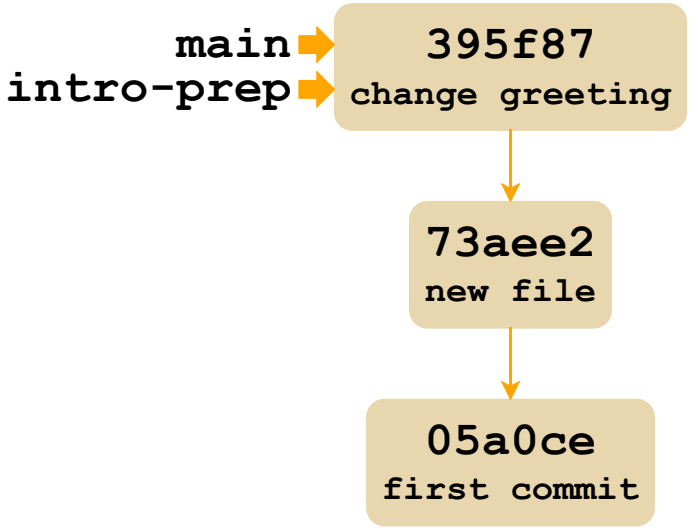
```
$ git branch intro-prep
```




```
$ tree .git
.git
├── ....
└── refs
    ├── heads
    │   ├── intro-prep
    │   └── main
    └── tags
```

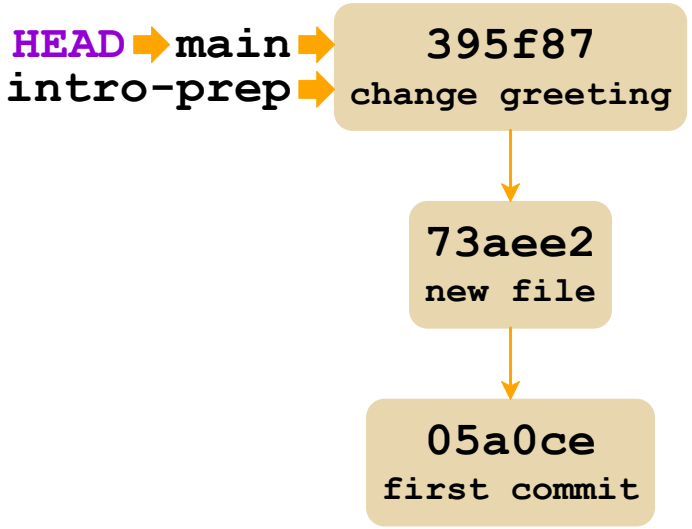
```
$ cat .git/refs/heads/intro-prep
395f876860c78f435ebf09b94372a35b1bebd01e
```

Branches as References to Commits



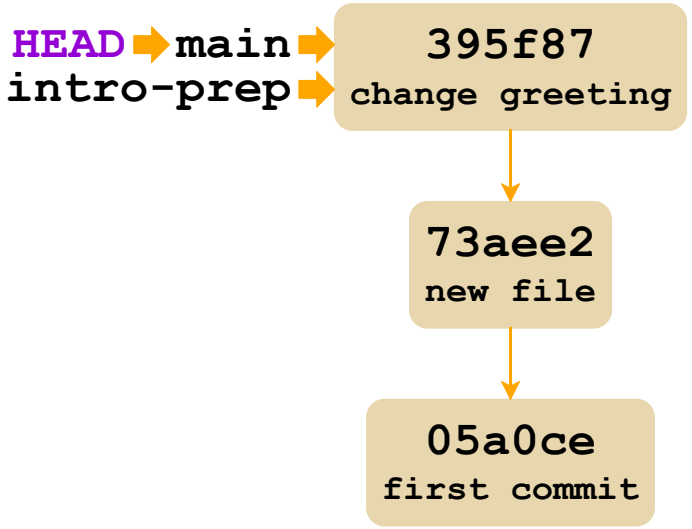

```
$ cat .git/HEAD
ref: refs/heads/main
```

Branches as References to Commits



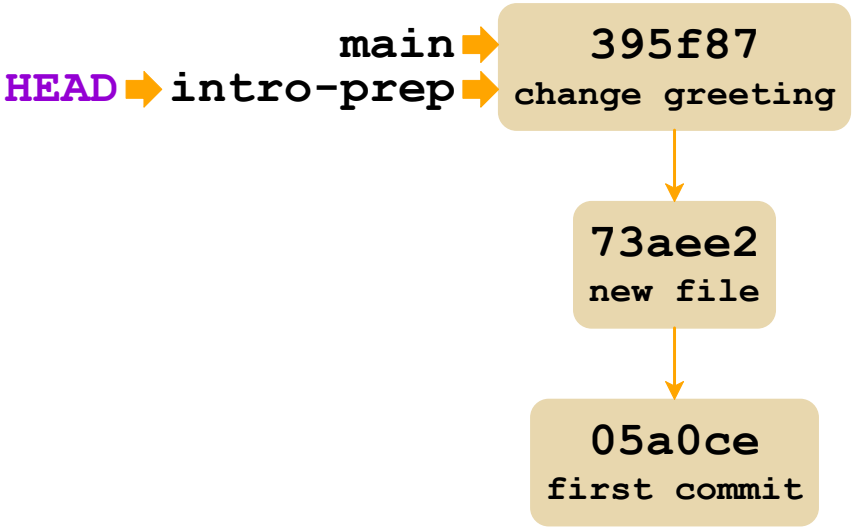

```
$ cat .git/HEAD
ref: refs/heads/main
```

Branches as References to Commits



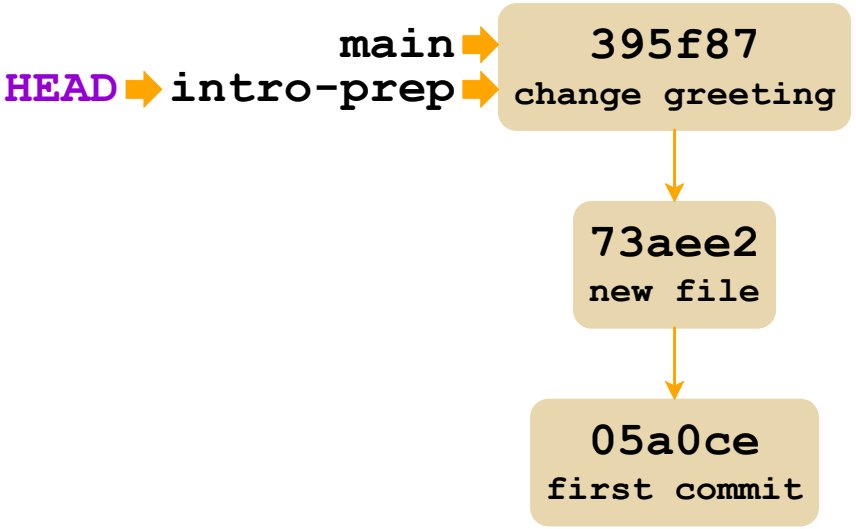

```
$ git branch
  intro-prep
* main
```

Branches as References to Commits




```
$ git checkout intro-prep

$ cat .git/HEAD
ref: refs/heads/intro-prep
```

Branches as References to Commits

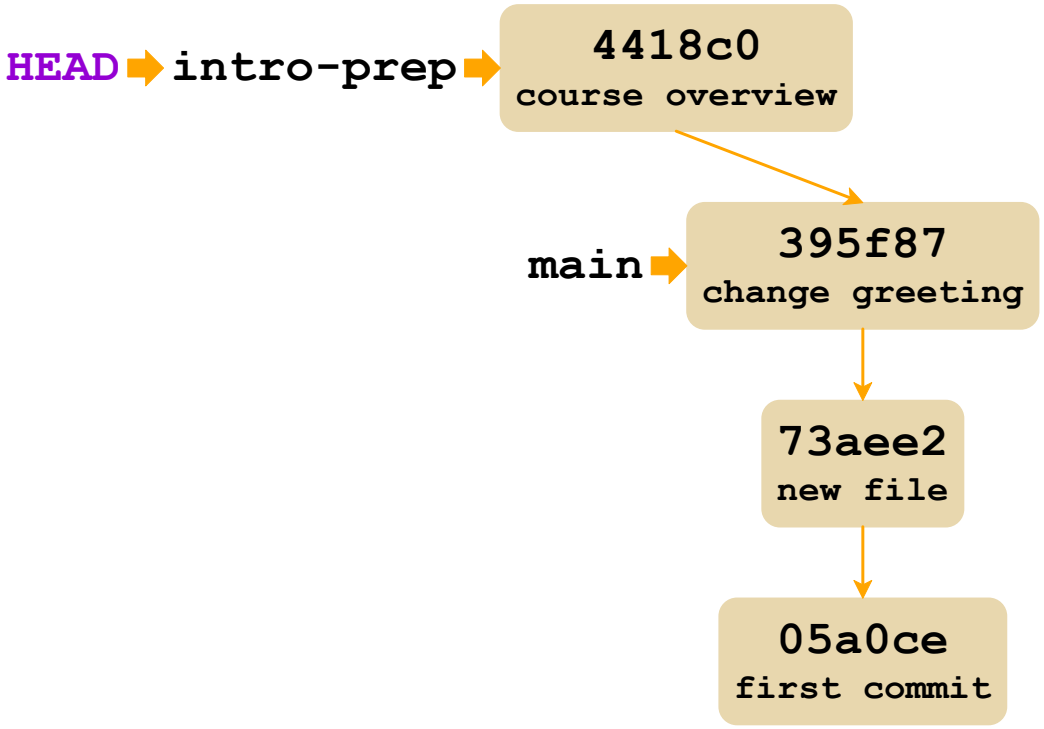



```
$ git branch
* intro-prep
  main
```

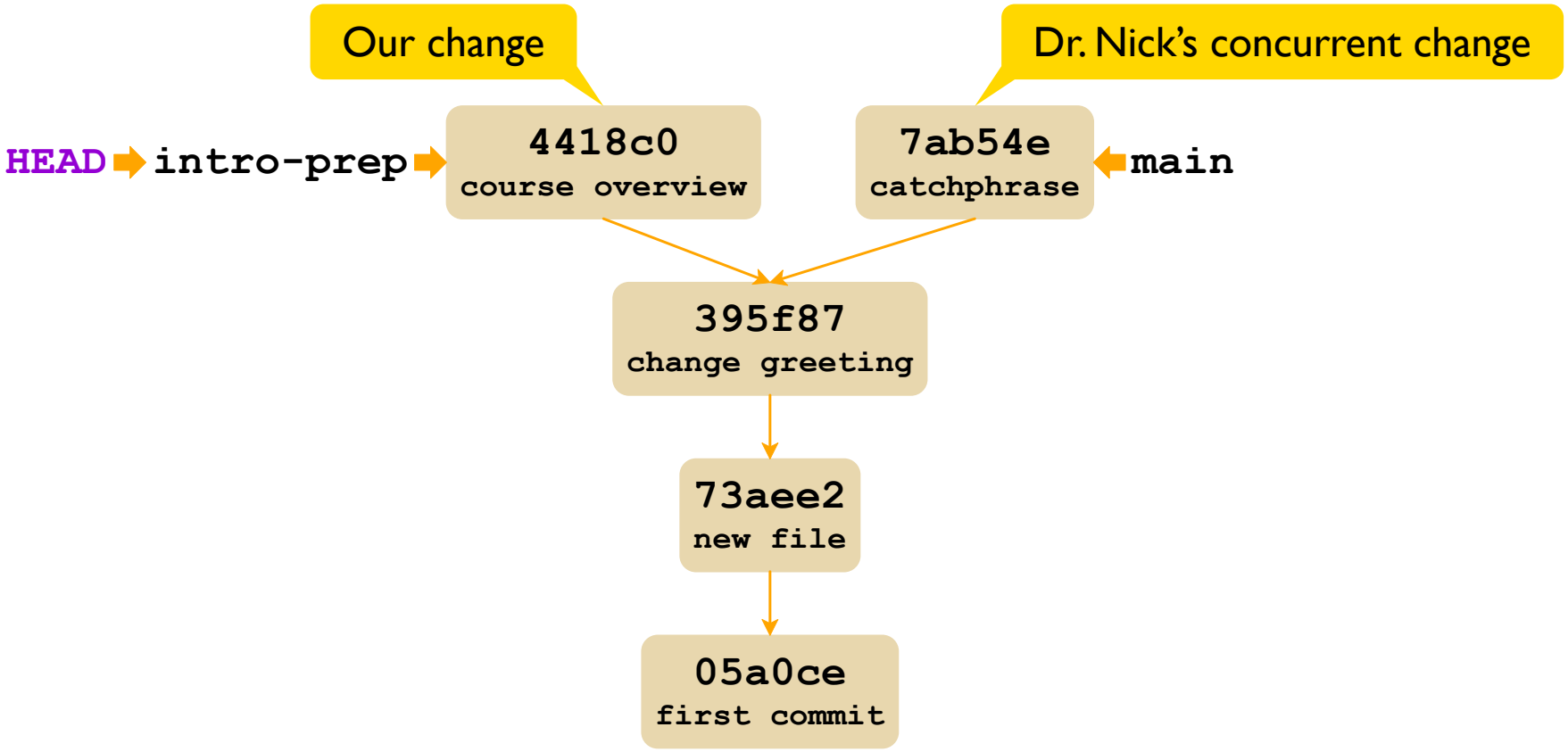
Changing a Branch

```
$ emacs greeting.txt
Hello, everyone!

Welcome to CS 6015 MSD SE.
A few simple ideas can vastly
improve software quality.

$ git add greeting.txt
$ git commit -m "course overview"
[intro-prep 4418c06] course overview
1 file changed, 4 insertions(+)
```

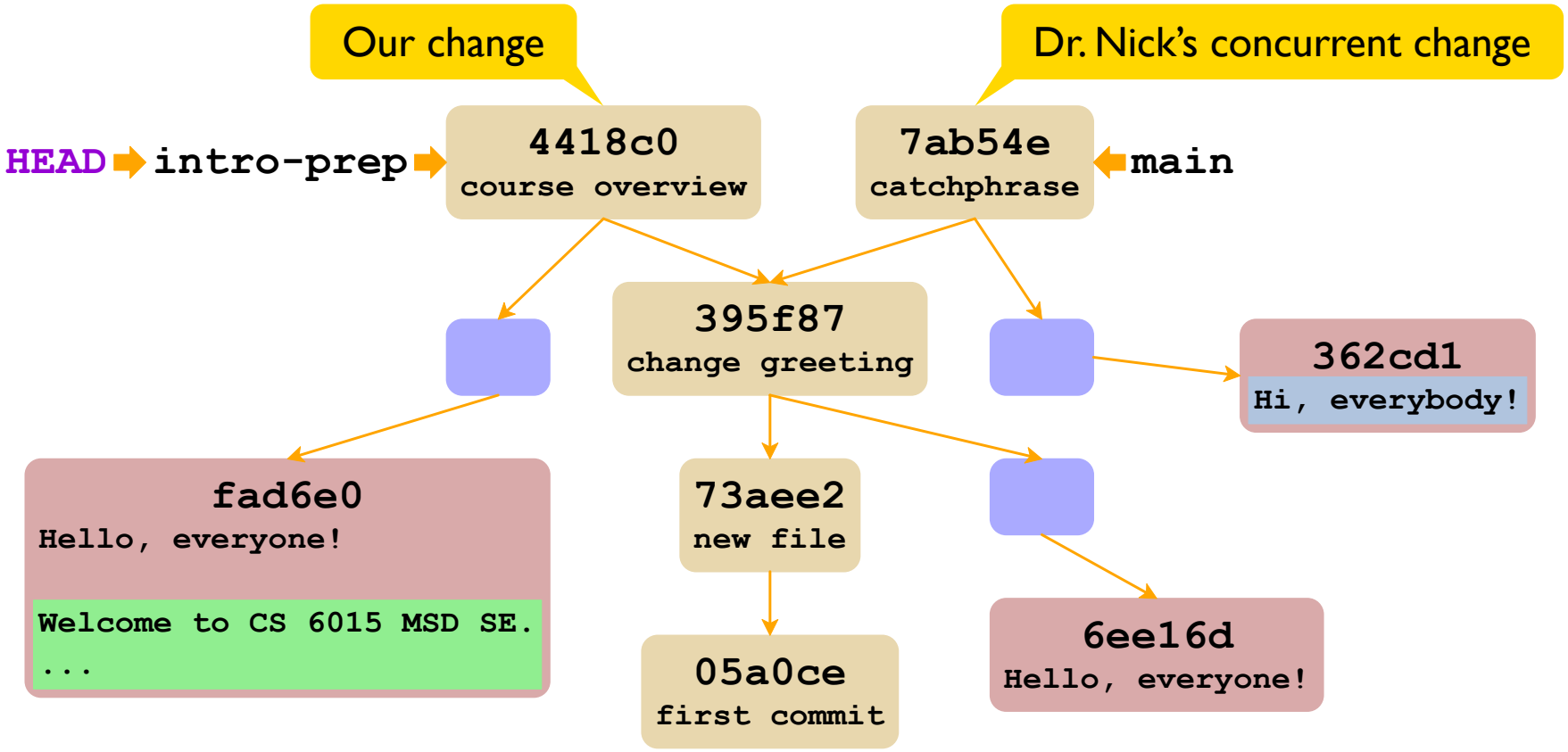
Store with Diverging Branches



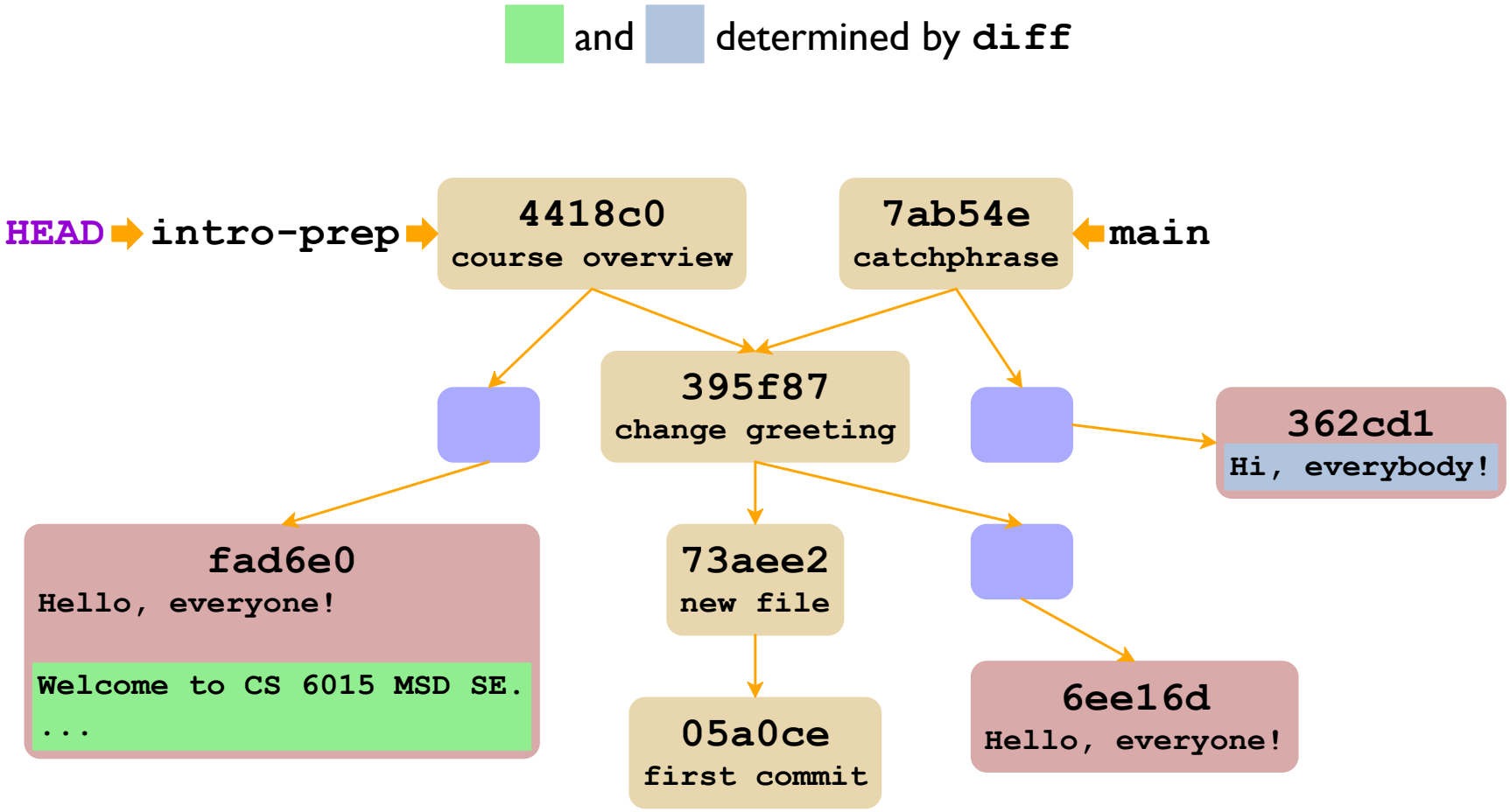
Store with Diverging Branches



Store with Diverging Branches

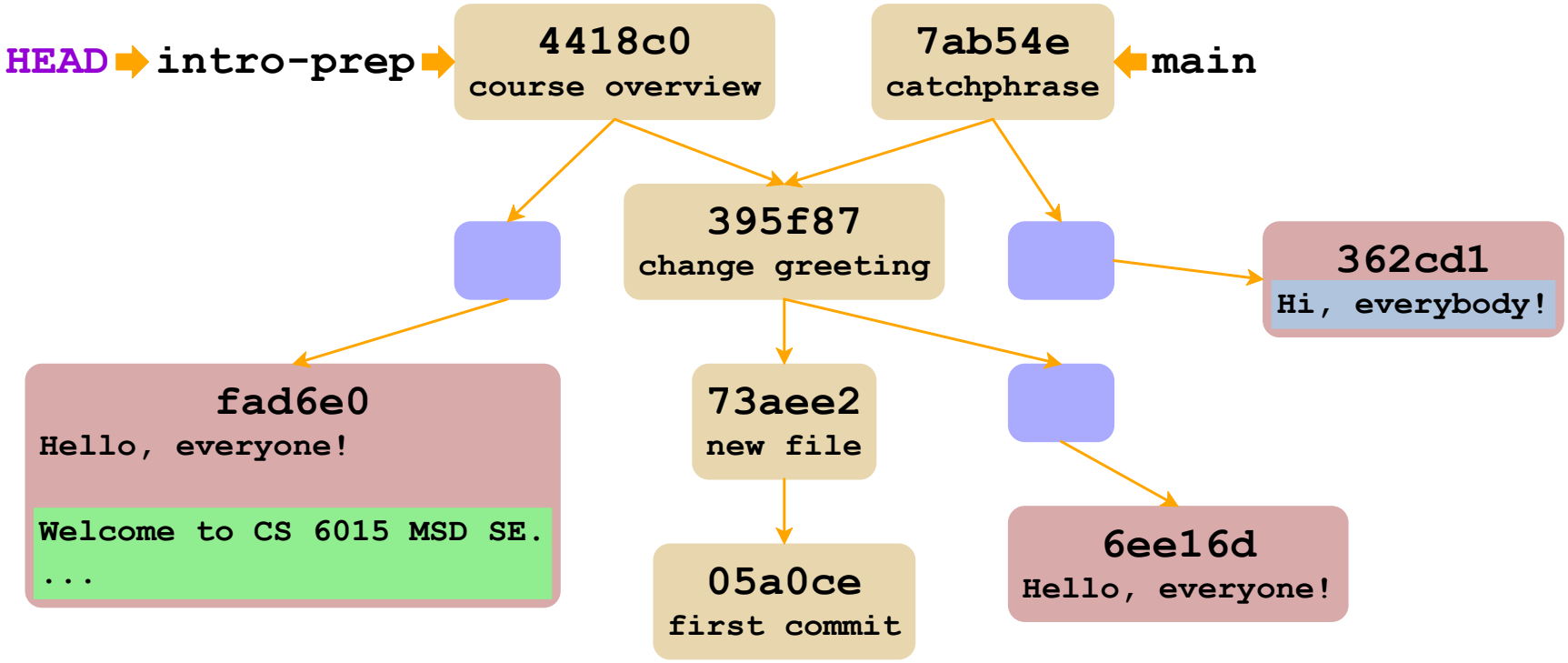


Store with Diverging Branches



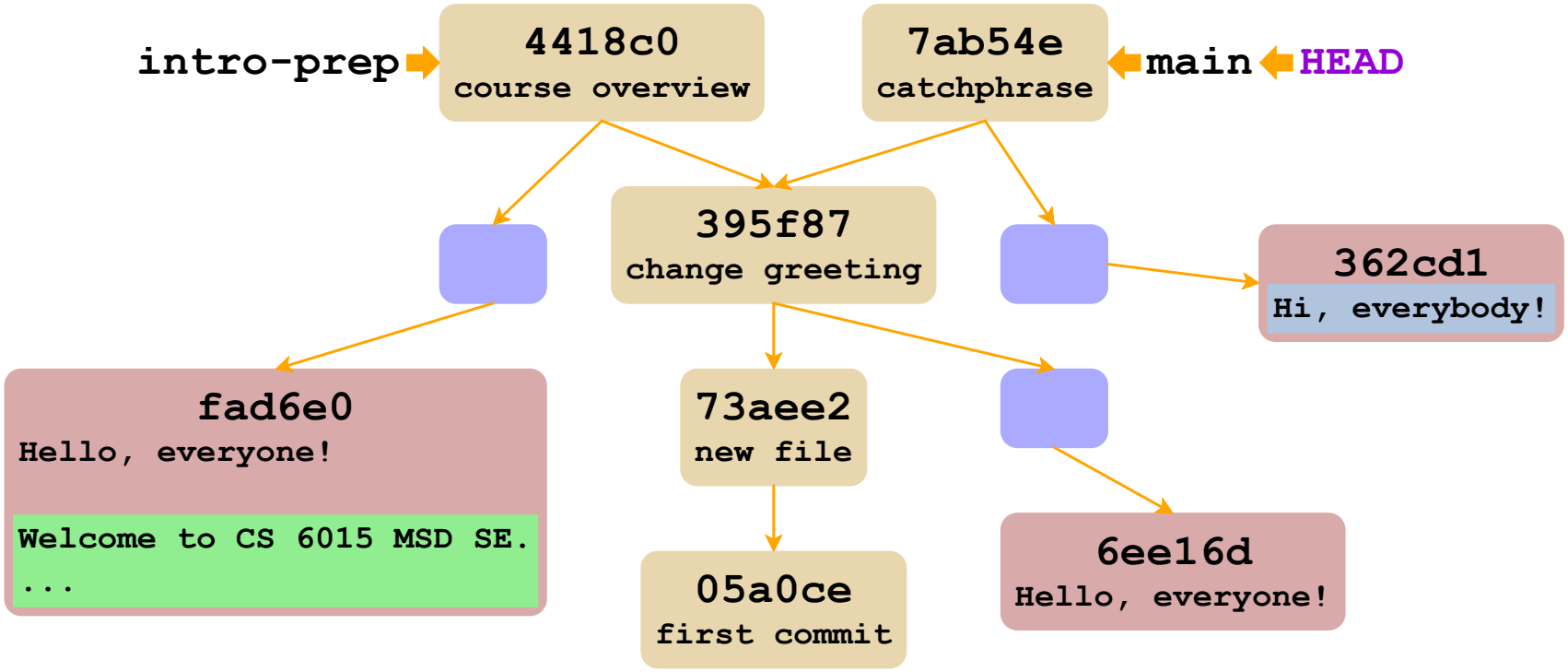
Store with Diverging Branches

```
$ git checkout main
```



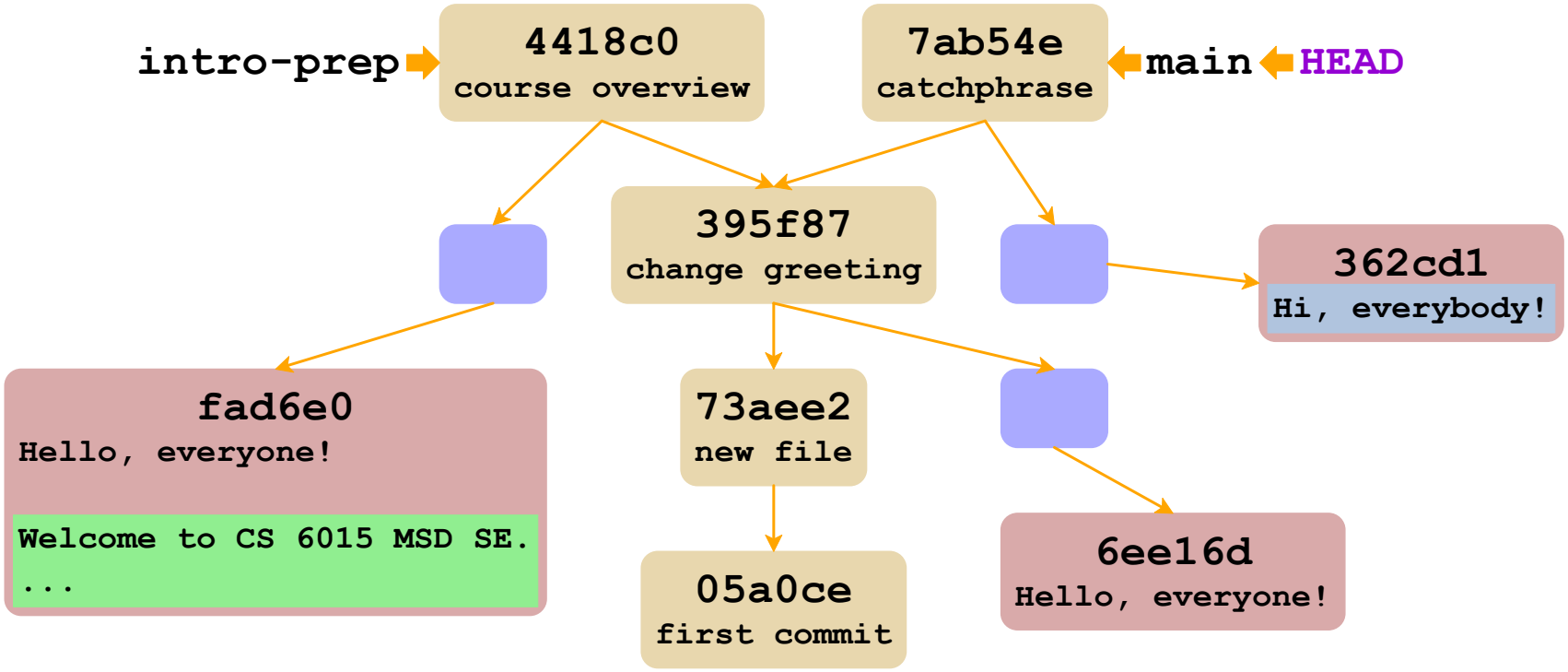
Store with Diverging Branches

```
$ git checkout main
```

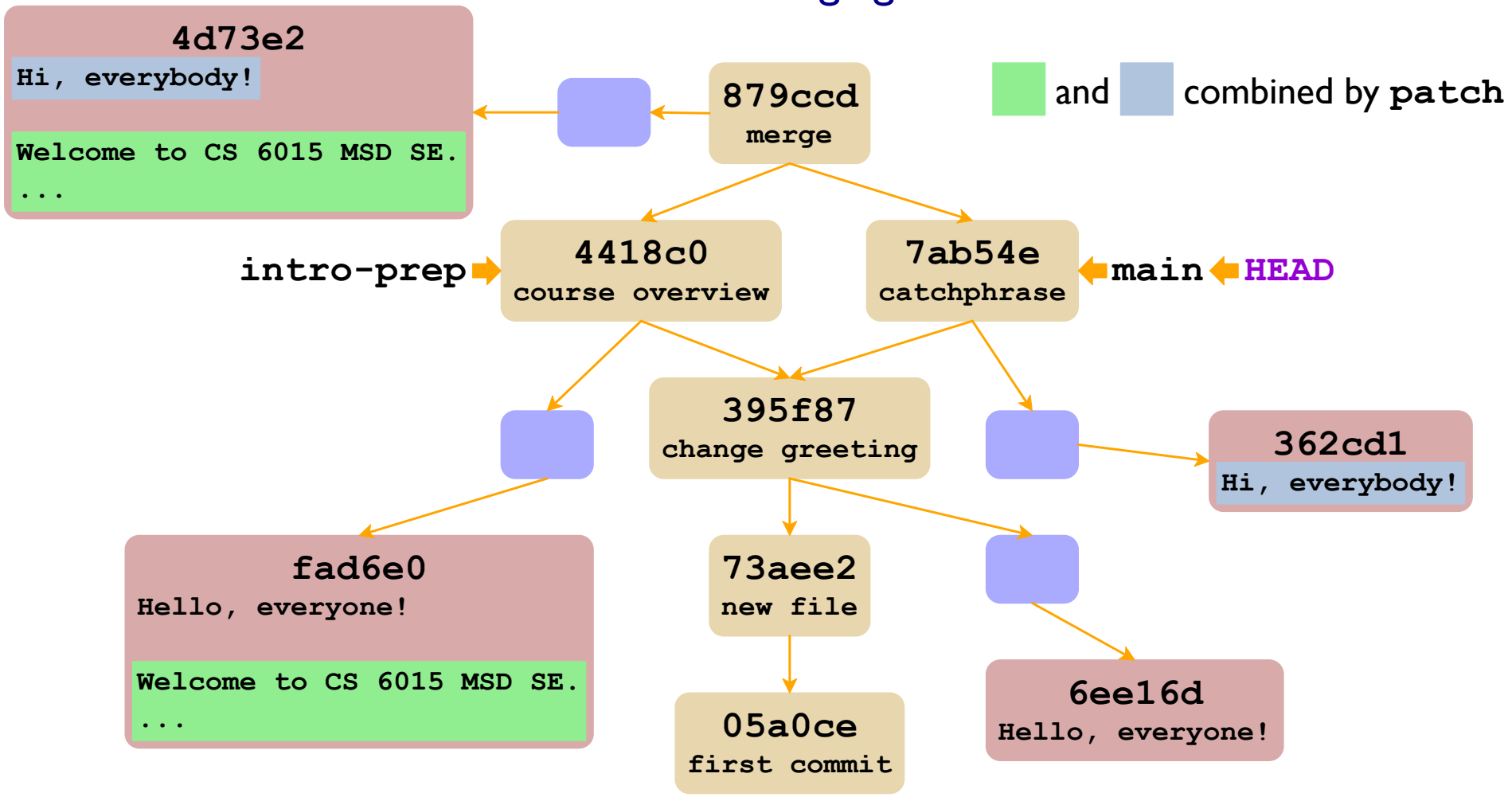


Store with Diverging Branches

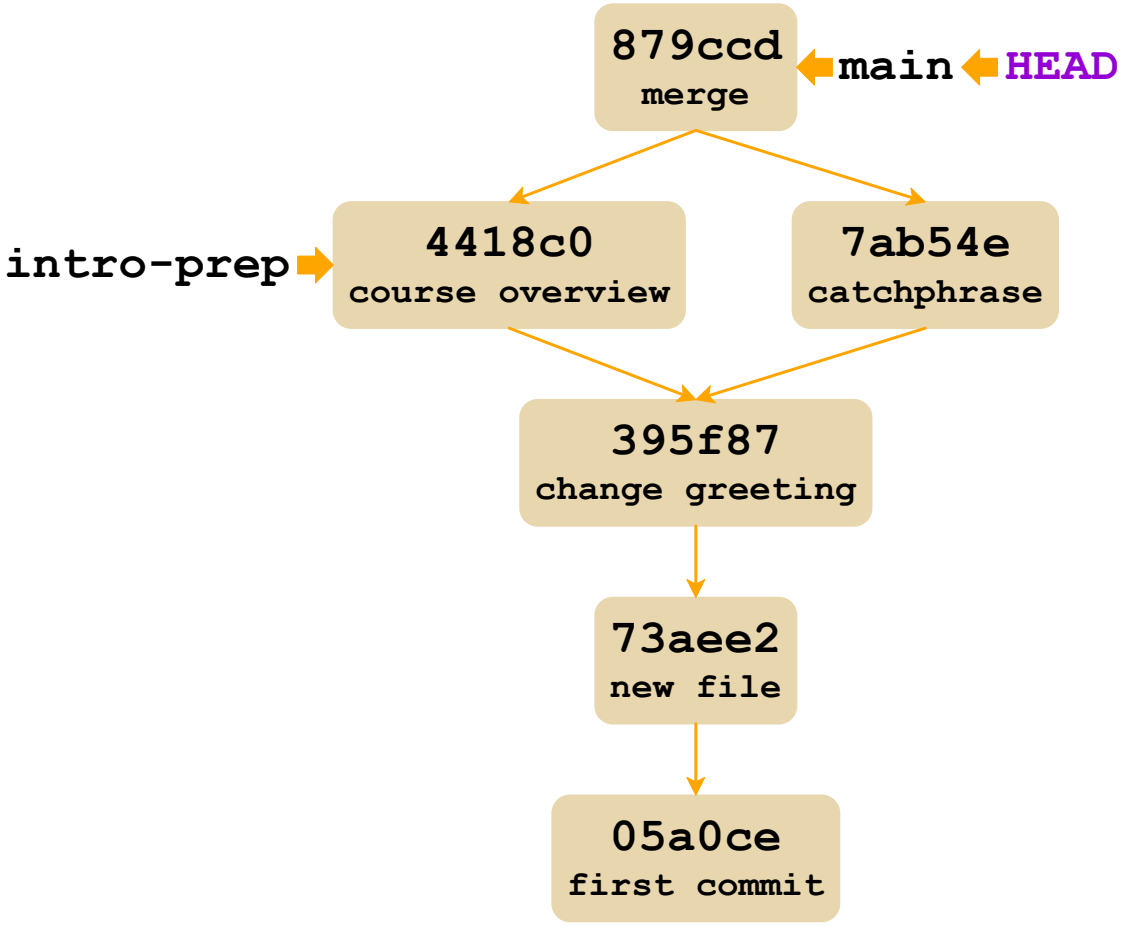
```
$ git merge intro-prep
```



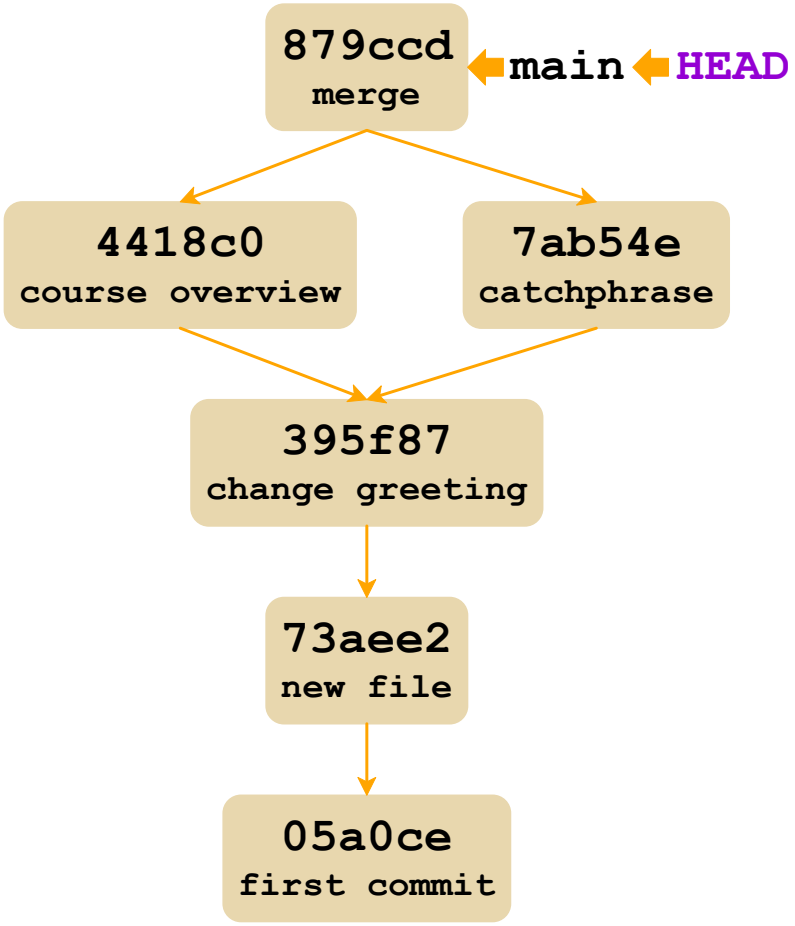
Store with Diverging Branches



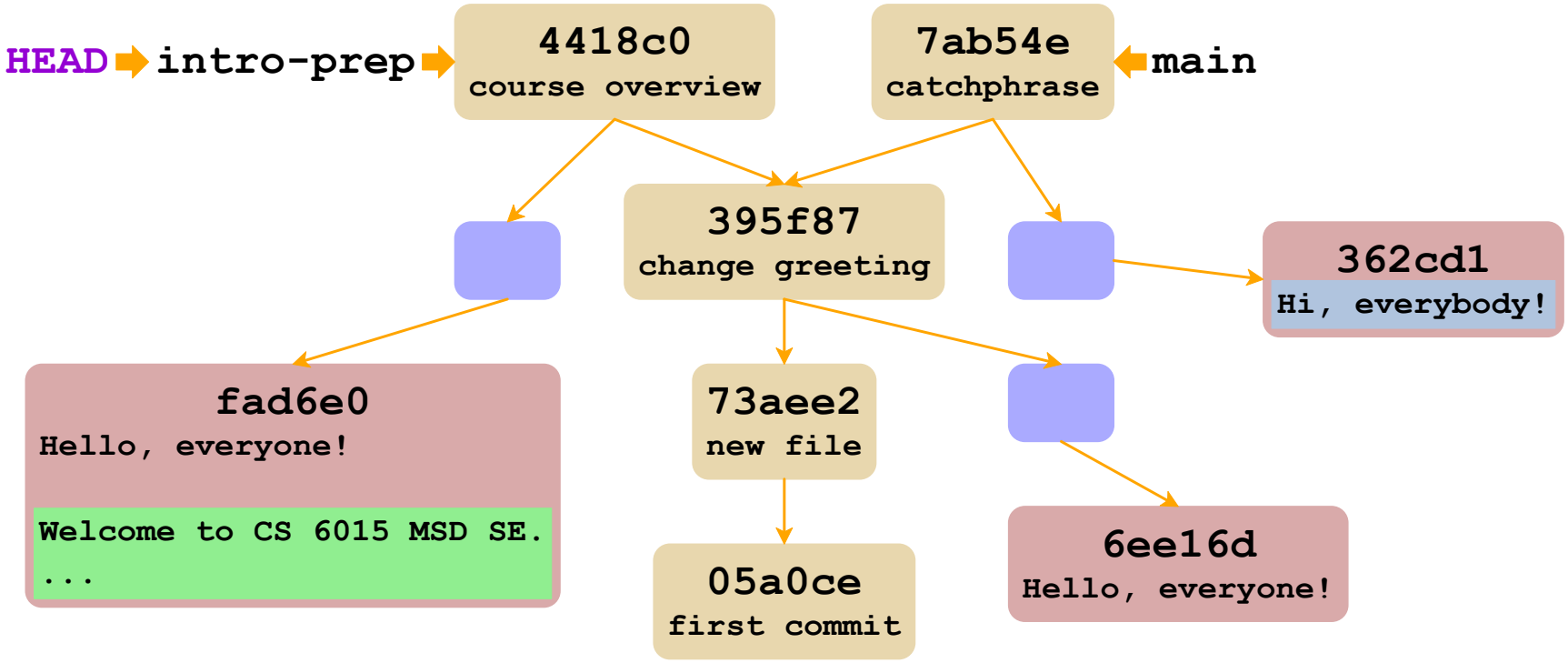
Store with Diverging Branches



Store with Diverging Branches

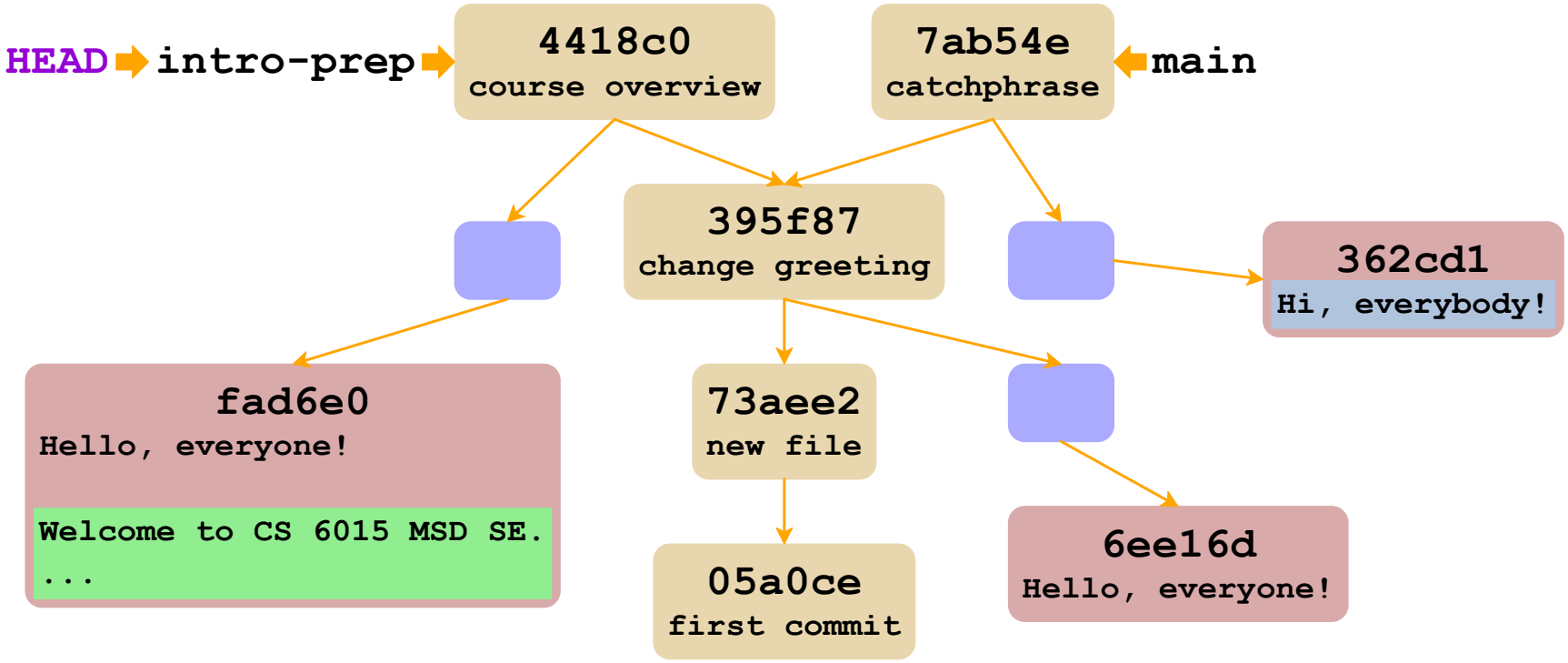


Merging with Rebase

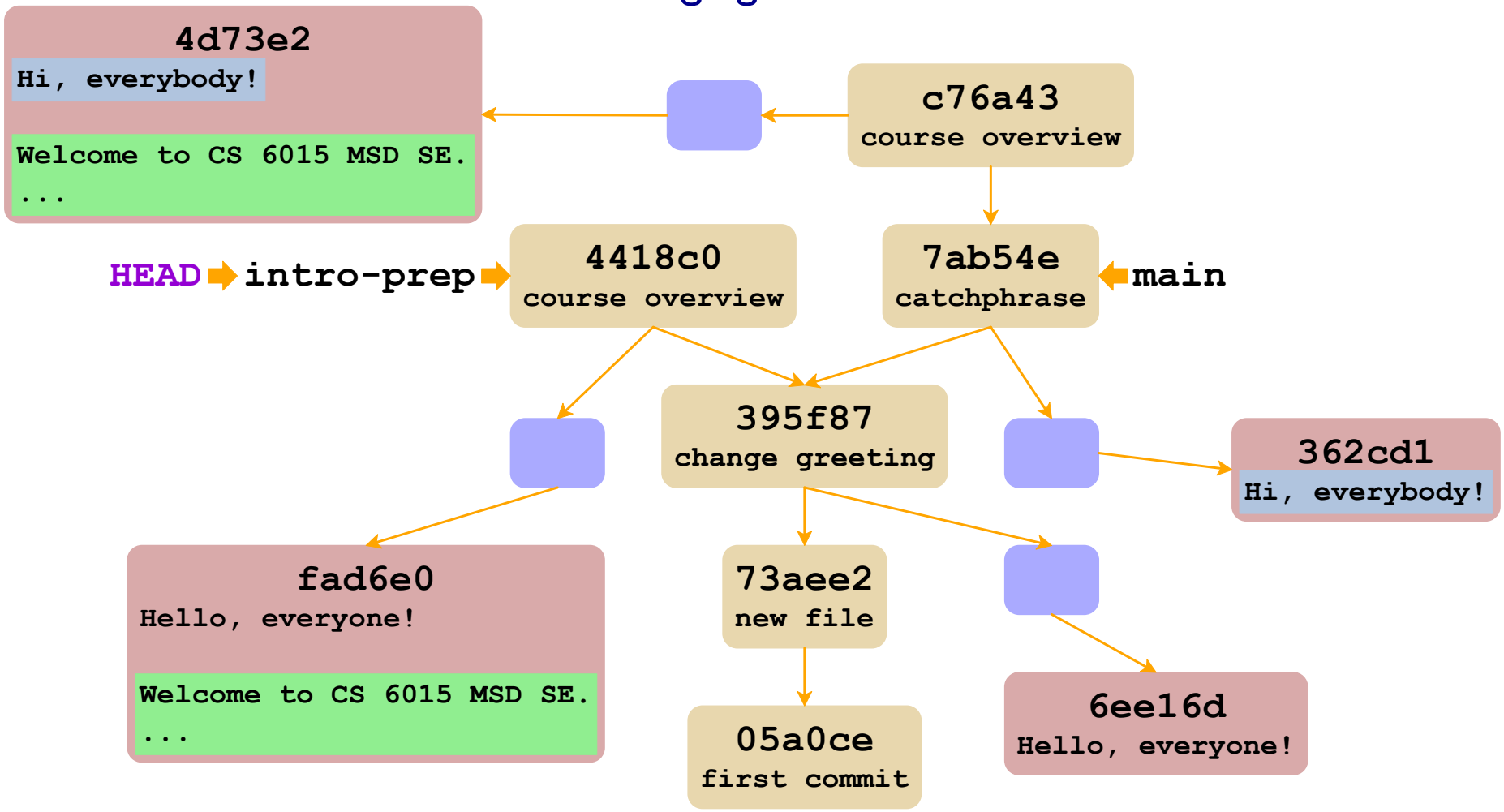


Merging with Rebase

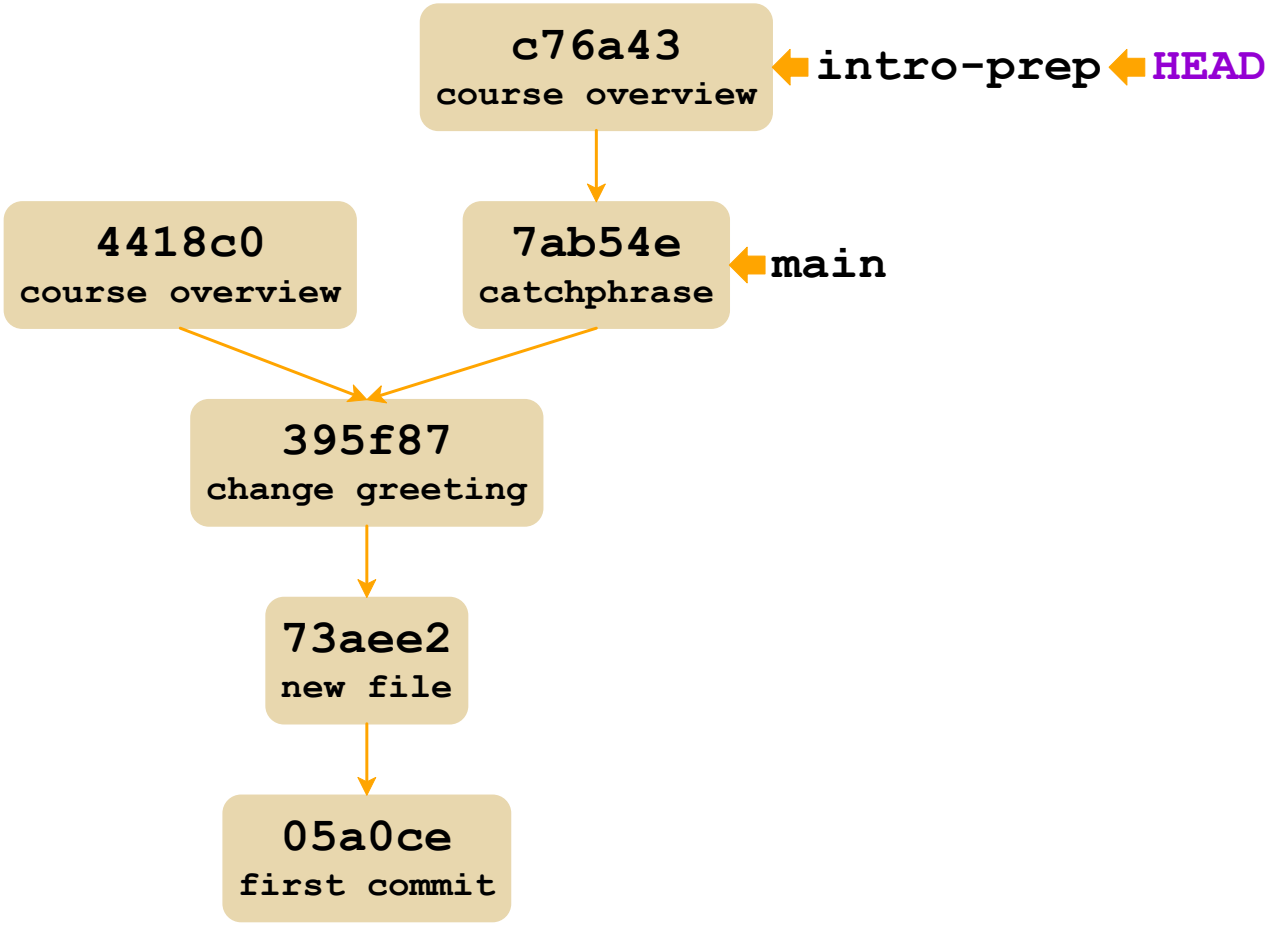
```
$ git rebase main
```



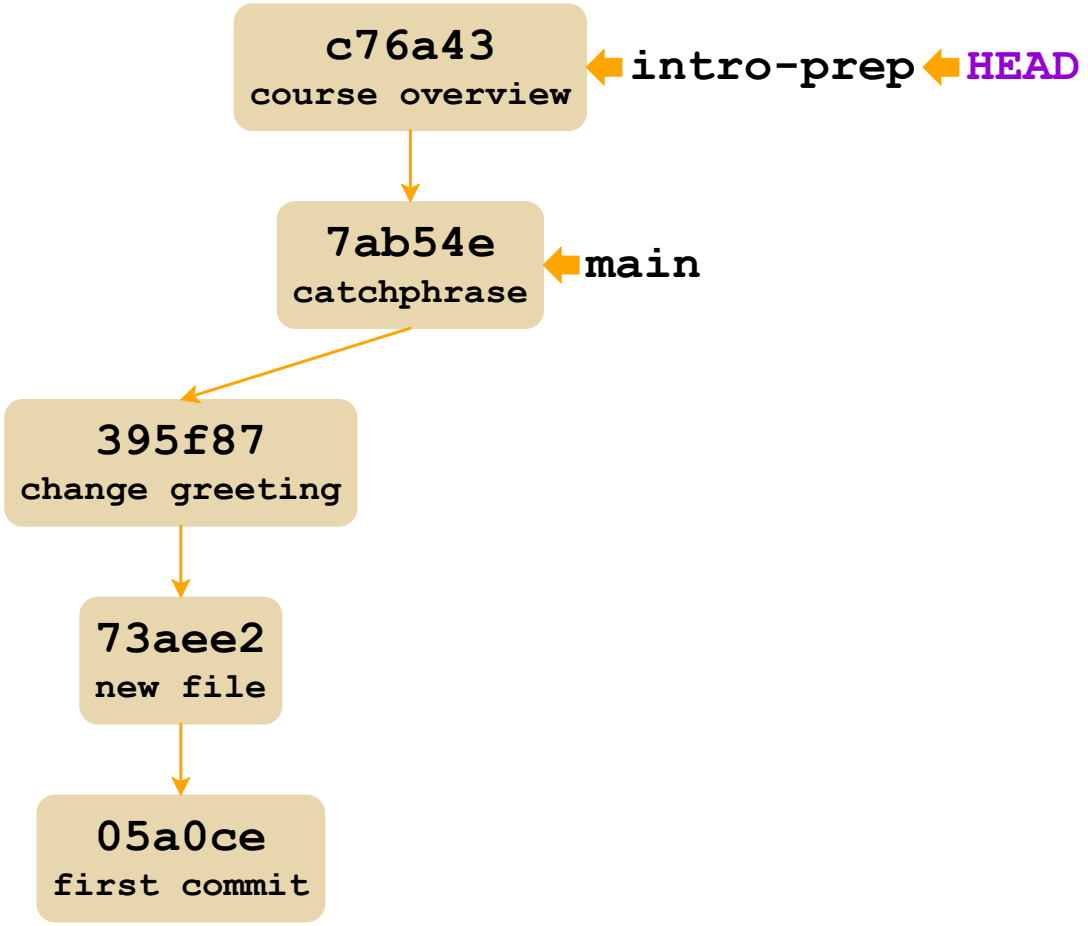
Merging with Rebase



Merging with Rebase



Merging with Rebase



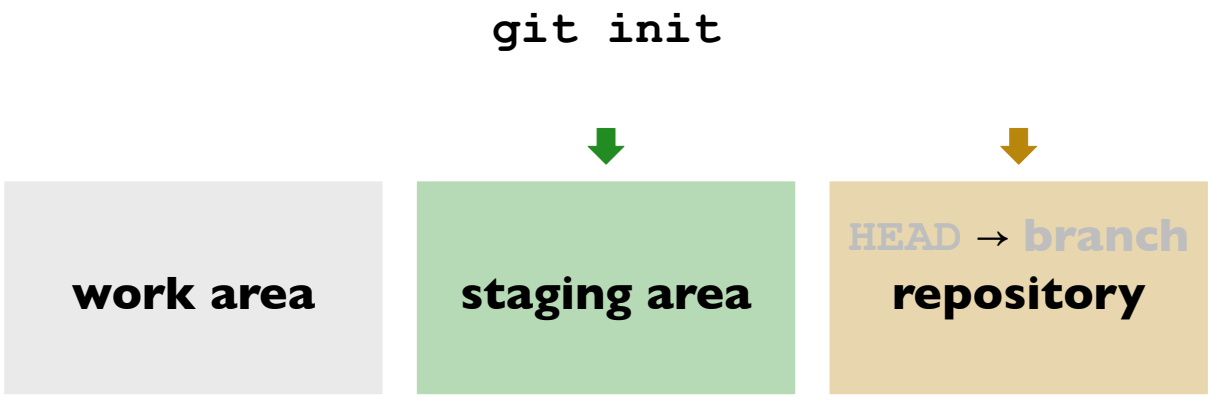
Using Git Branches Effectively

It's really pretty simple, just think of branches as references to hash-indexed immutable nodes in an object store.

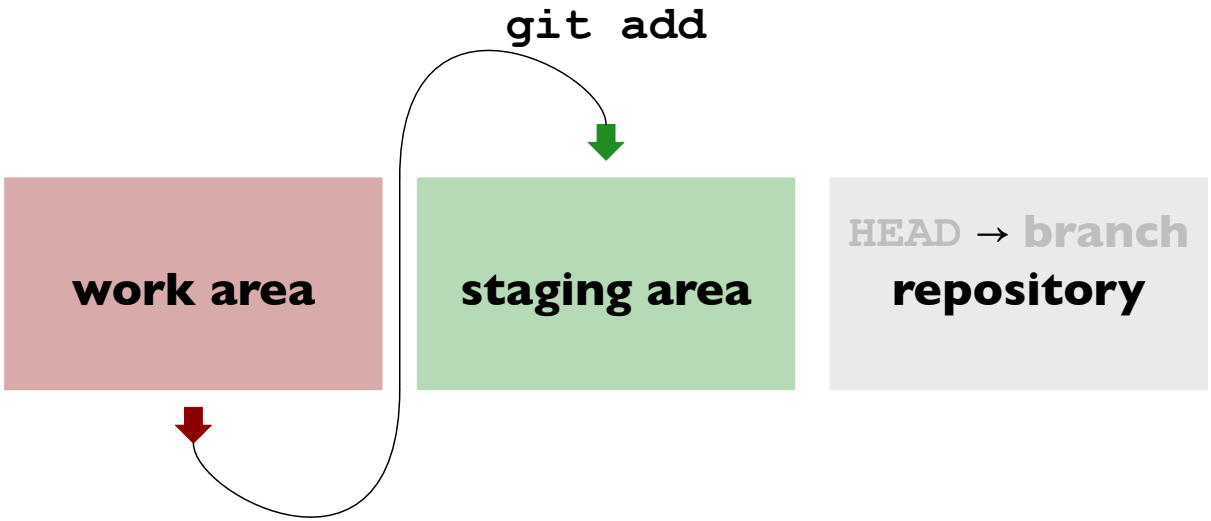
- Branches are cheap; create them freely
- Know what each Git command modifies
  - **work area**
  - **staging area**
  - **branch commits**
  - **`HEAD`**
- Be willing to manipulate the graph (e.g., rebase)

Part 4: Survey of Git Commands

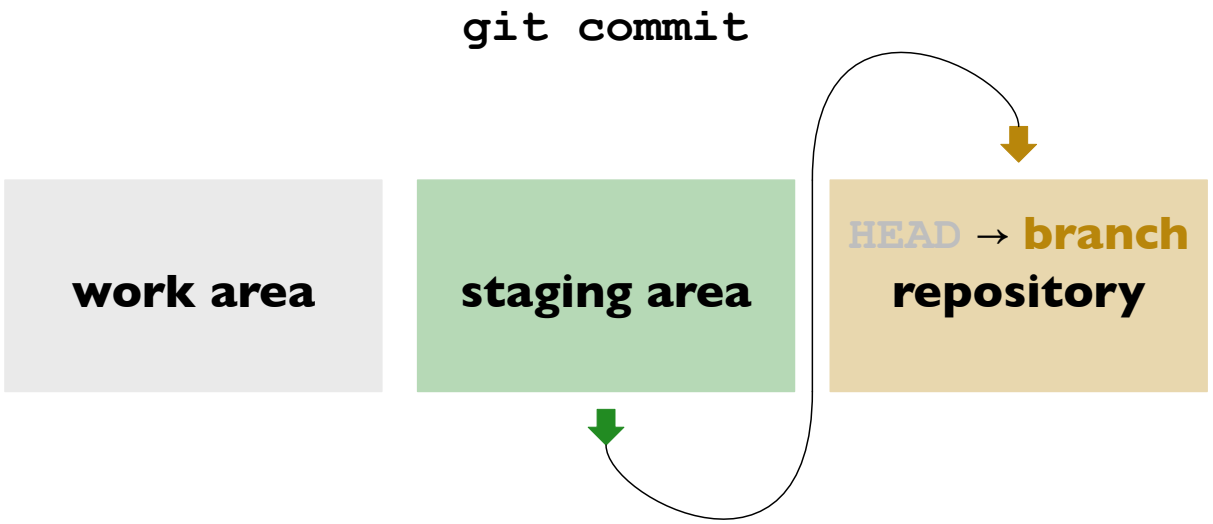
Git Command Survey



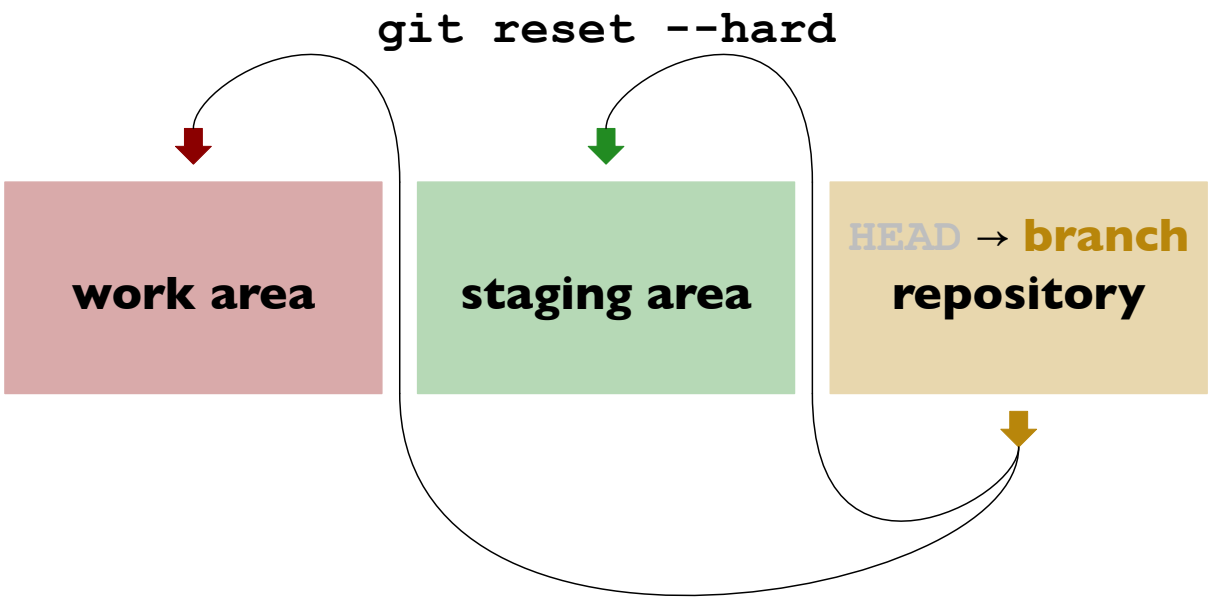
Git Command Survey



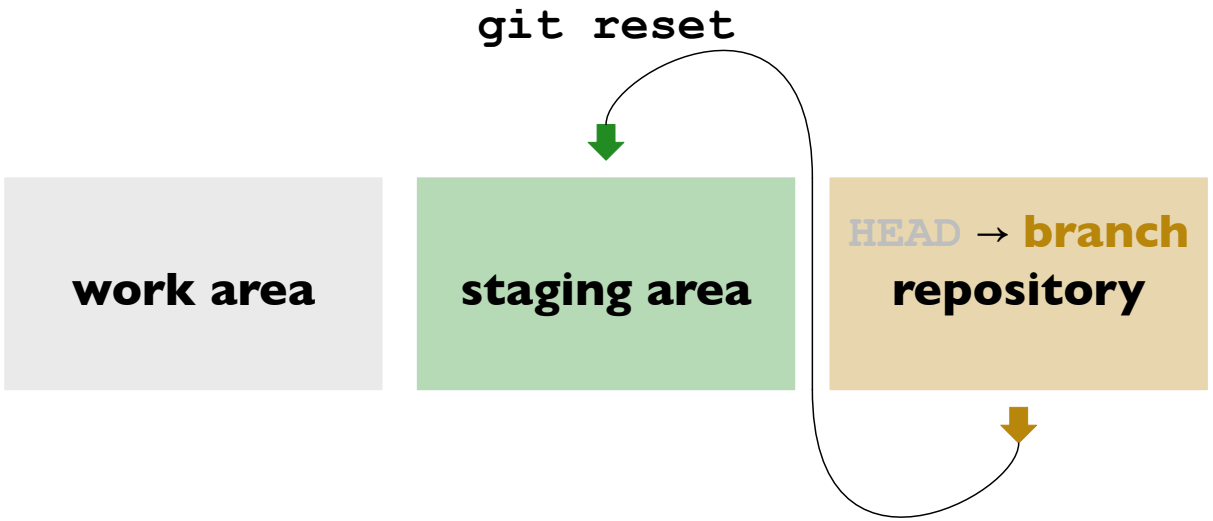
Git Command Survey



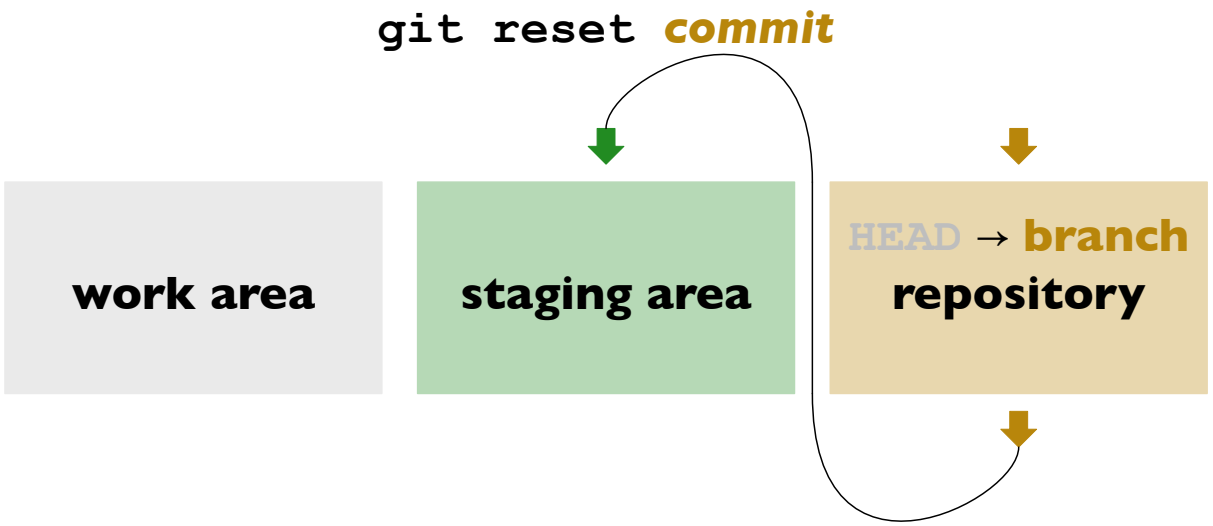
Git Command Survey



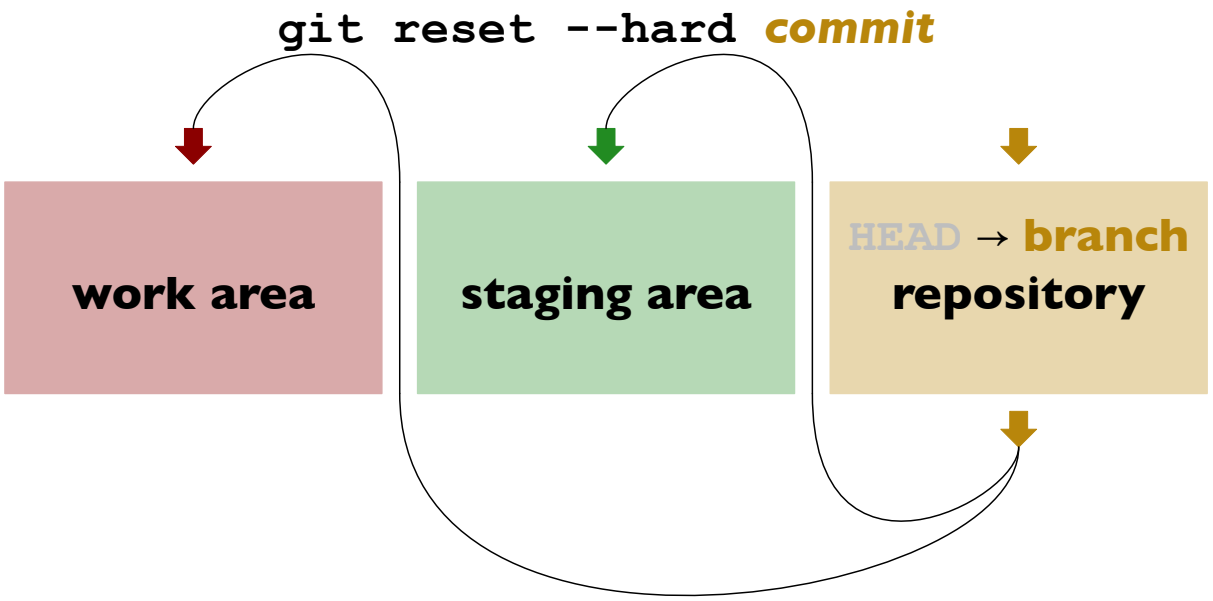
Git Command Survey



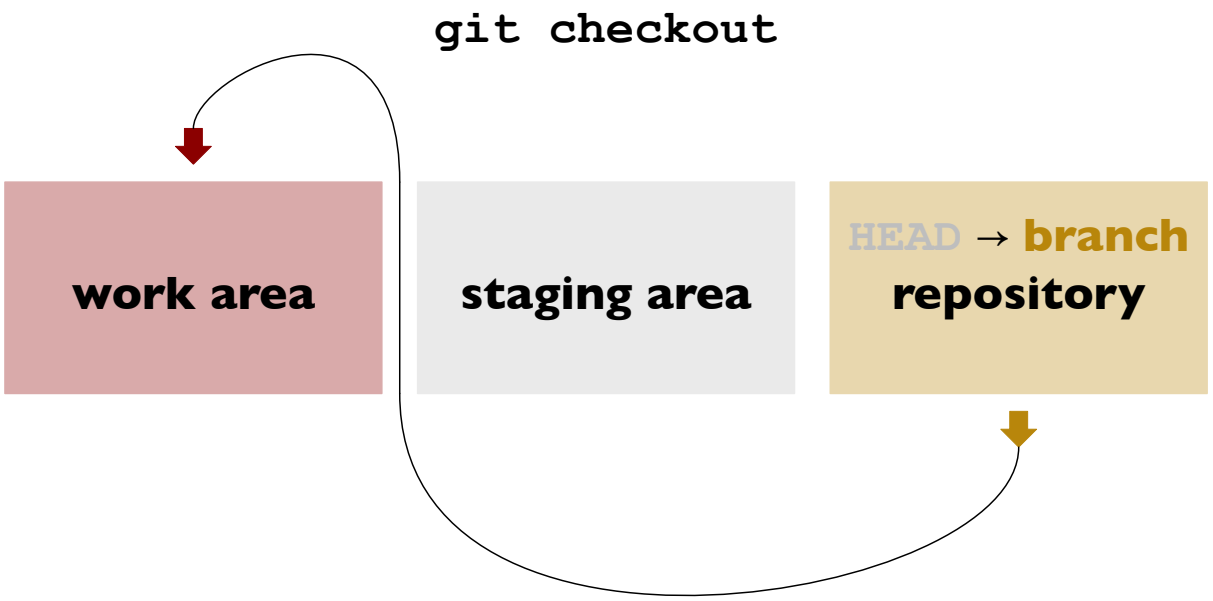
Git Command Survey



Git Command Survey

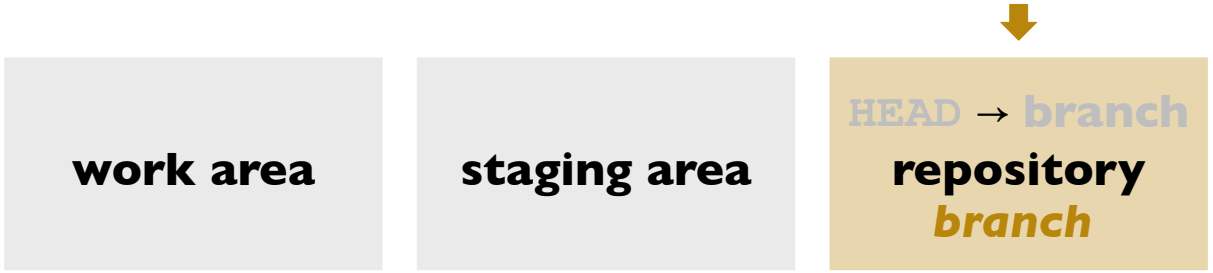


Git Command Survey

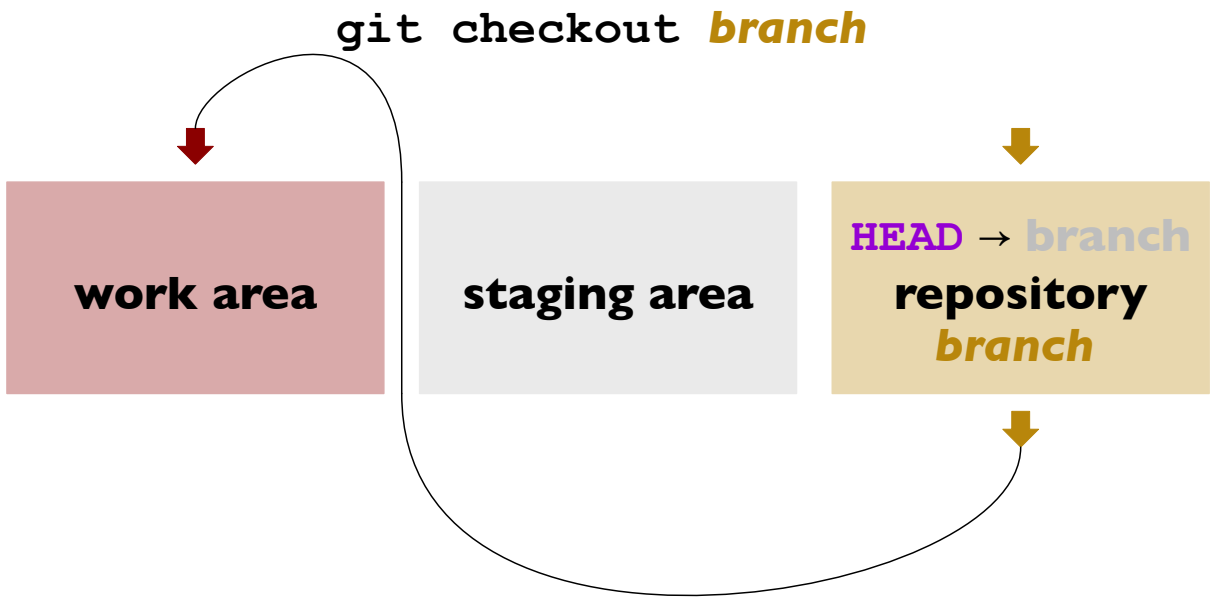


Git Command Survey

`git branch` ***branch***

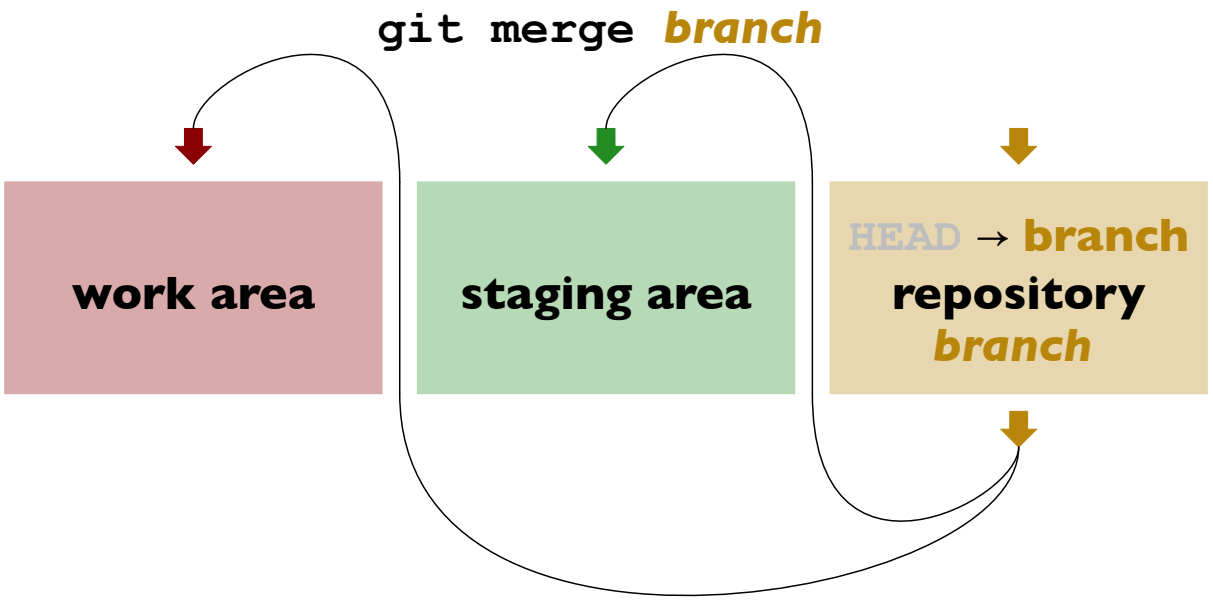


Git Command Survey



If possible while keeping **work area** and **staging area** changes!

Git Command Survey

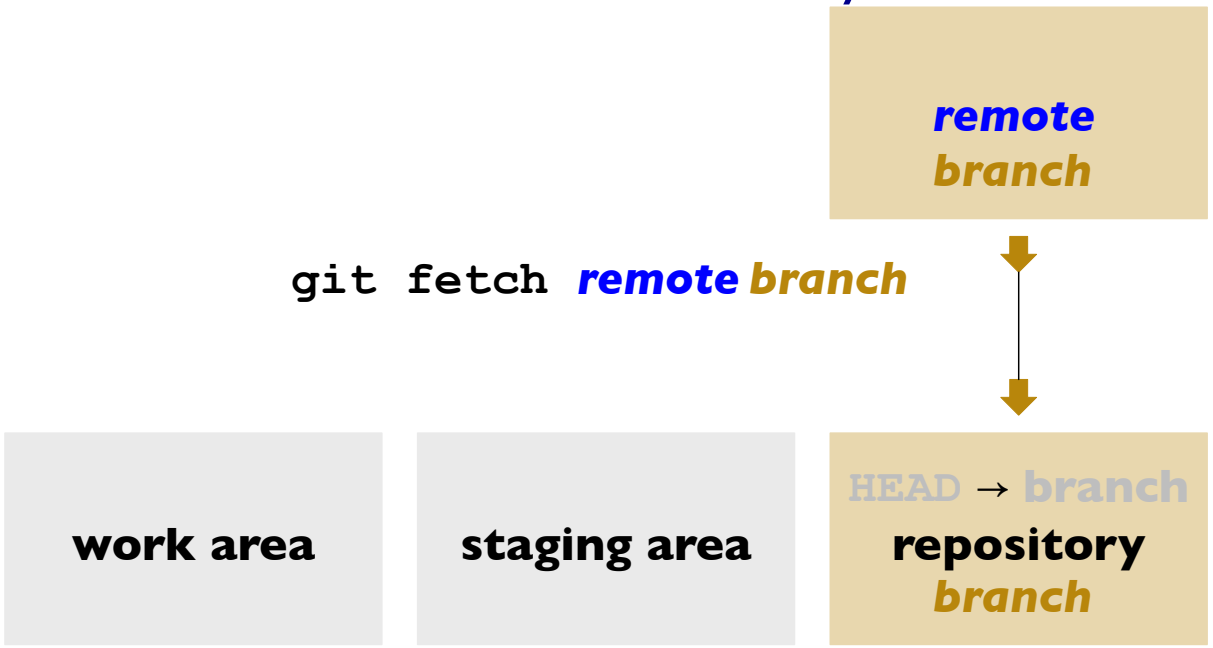


Part 4: Pull and Push

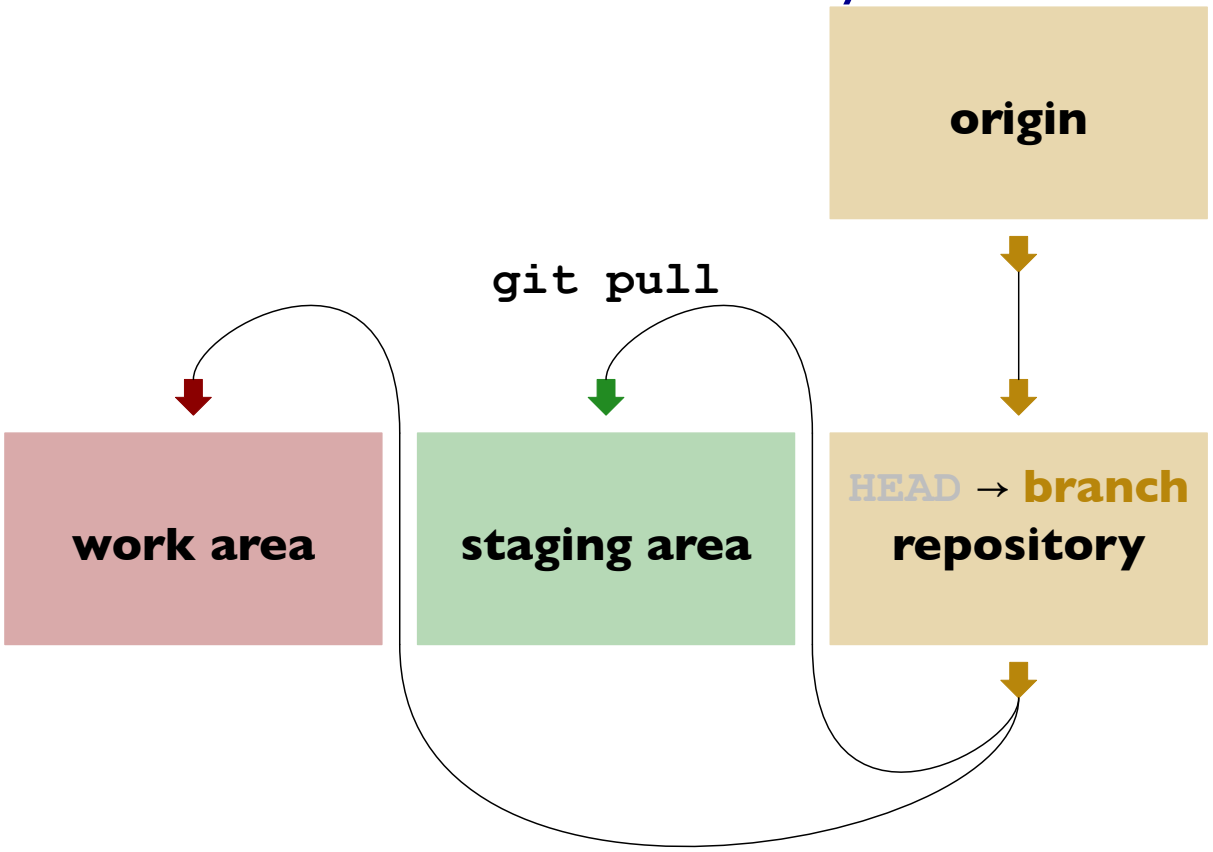
Git Command Survey

Git Command Survey

Git Command Survey

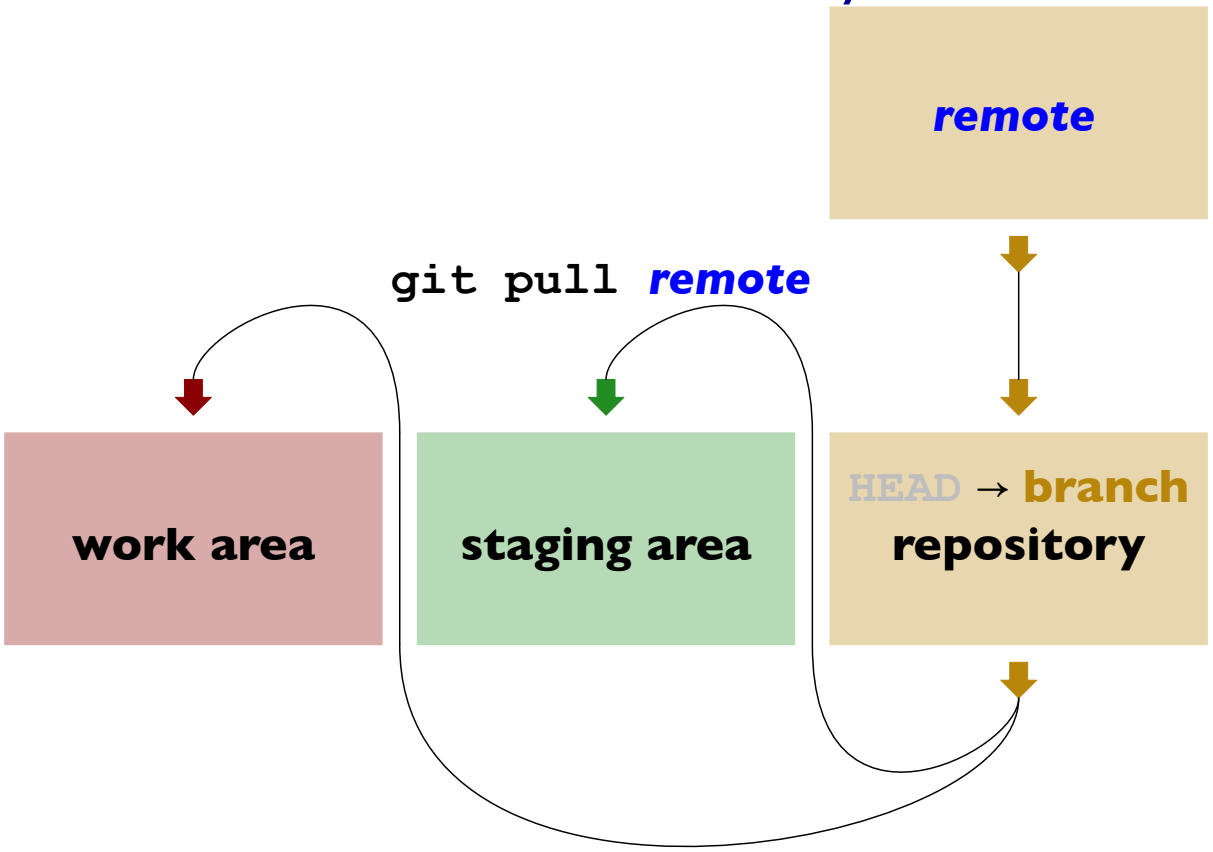


Git Command Survey



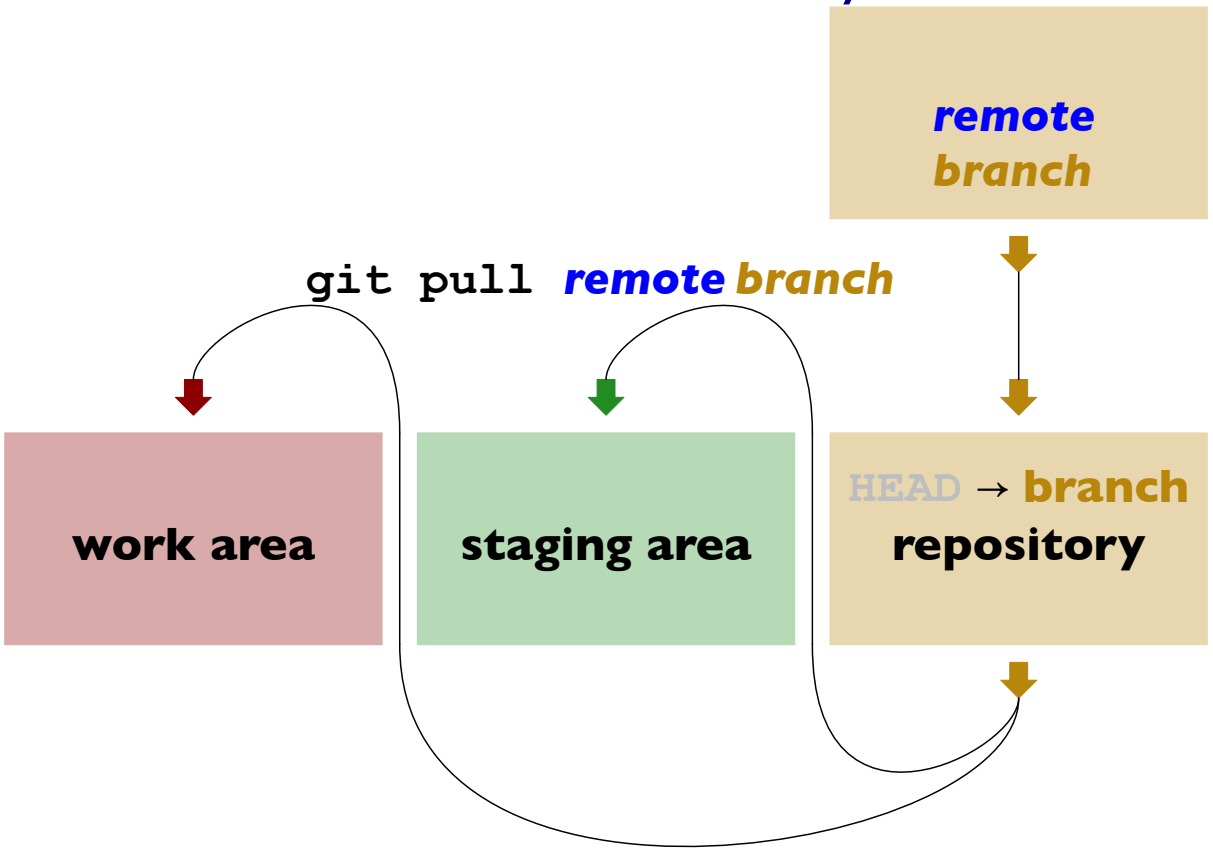
`git pull` is essentially `git fetch && git merge`

Git Command Survey



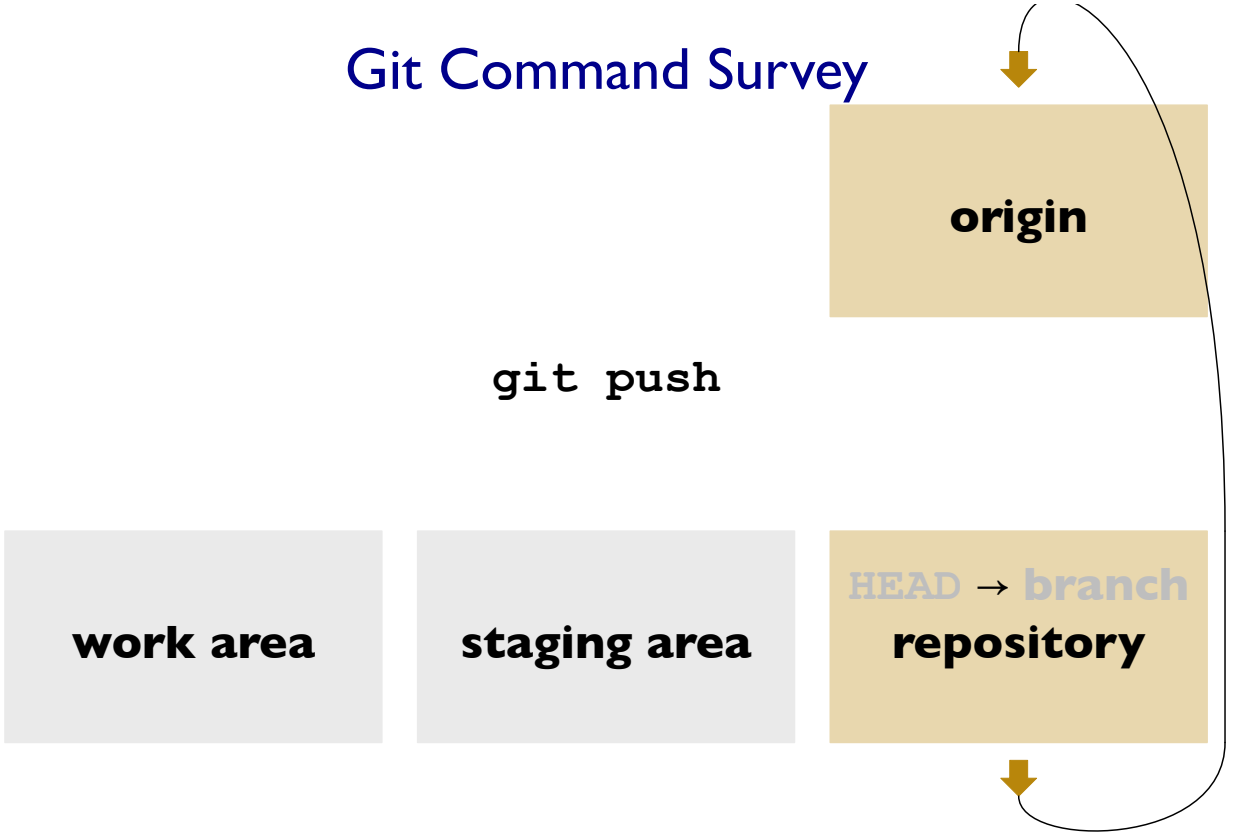
`git pull` is essentially `git fetch && git merge`

Git Command Survey

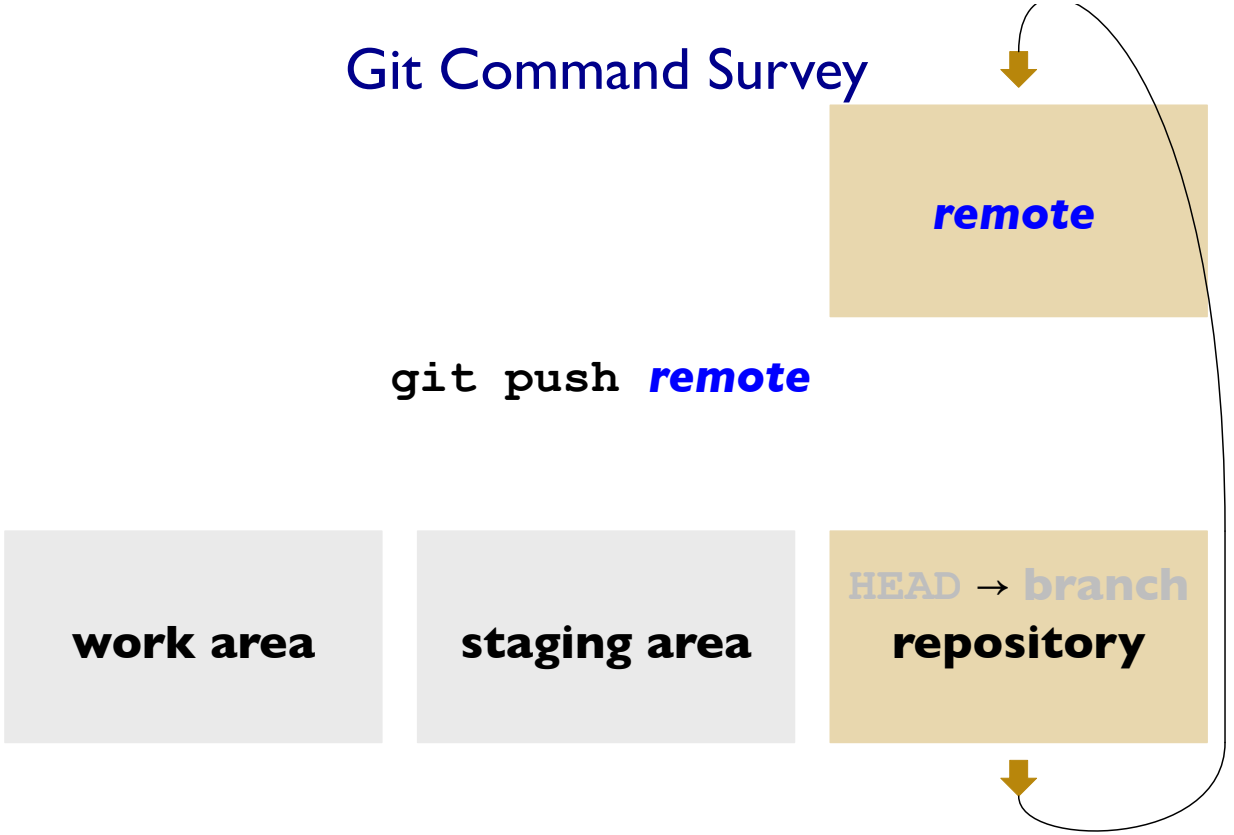


`git pull` is essentially `git fetch && git merge`

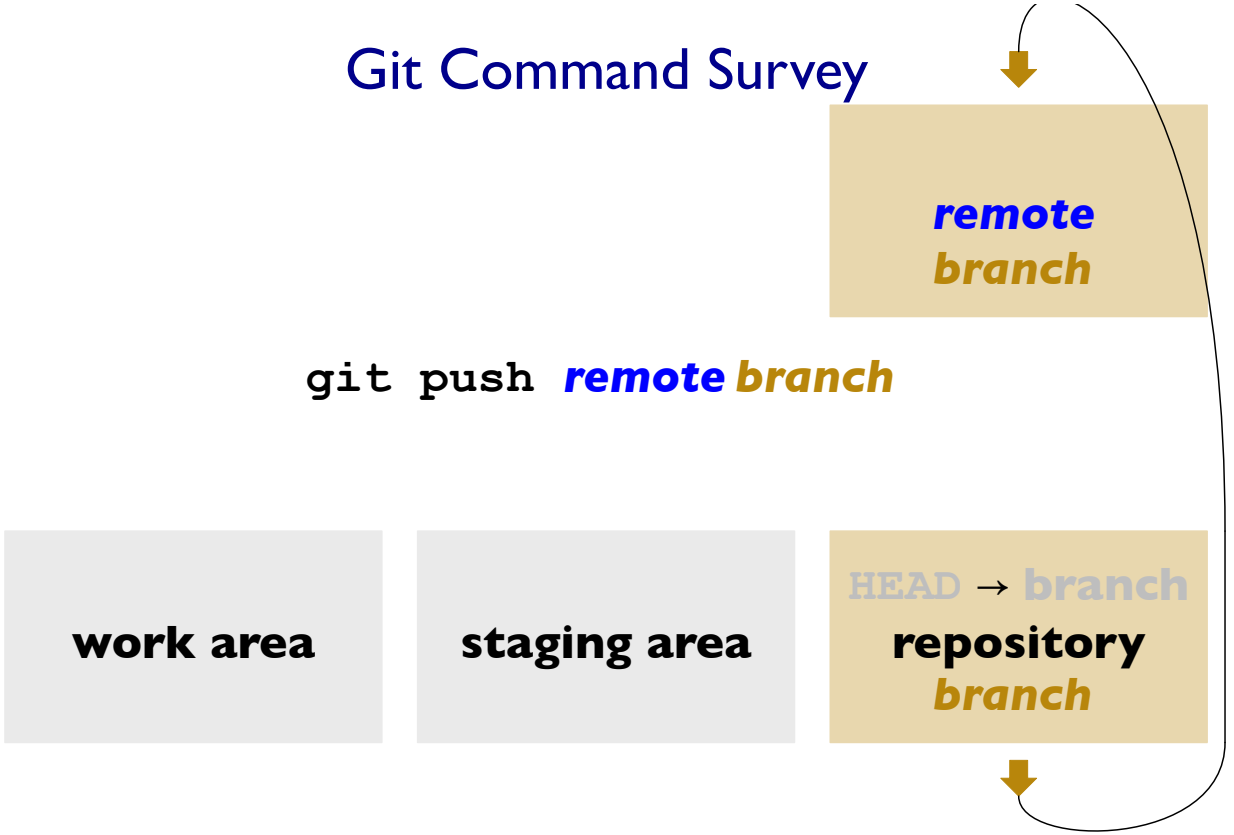
Git Command Survey



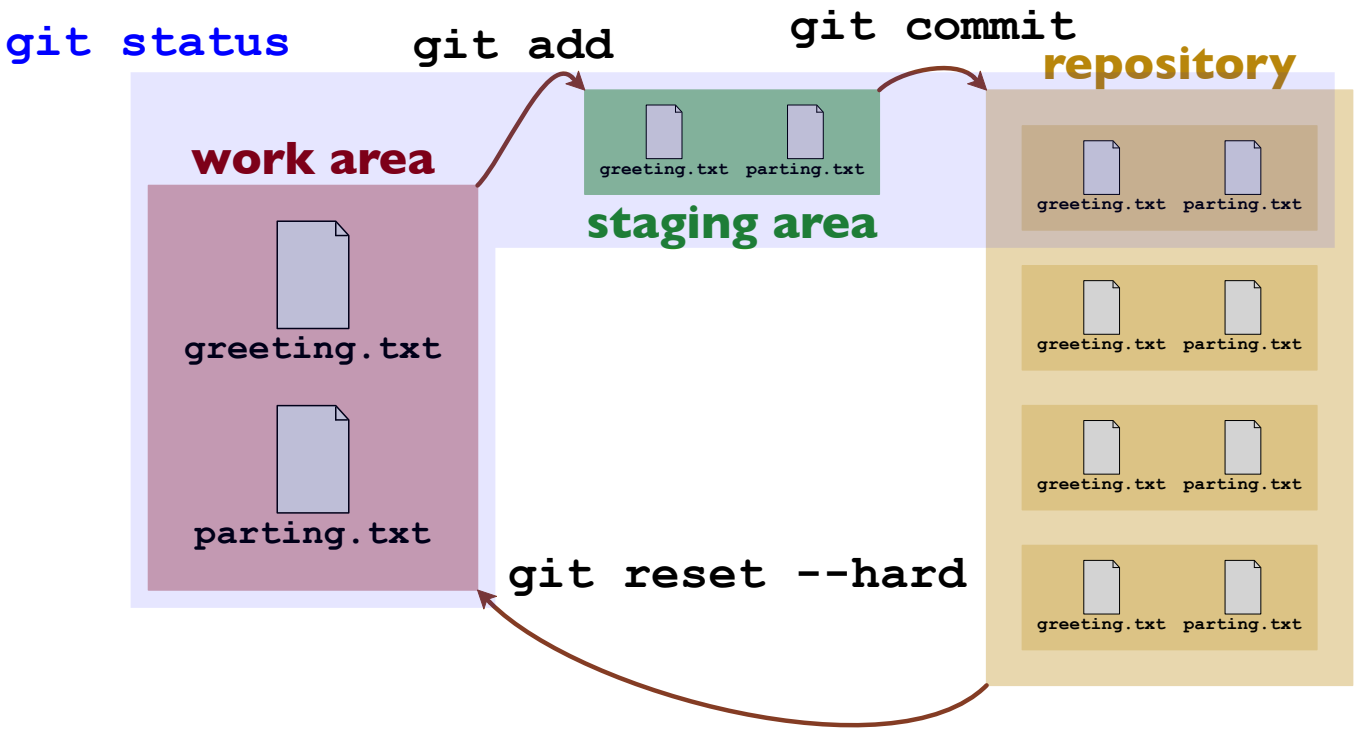
Git Command Survey



Git Command Survey



Summary



73aee2937a7fe941d79c855386e599739fe89b15