BashLand Course

Prepared by: Oleksii Stroganov

A 4-hour intensive on bash & git
using a real bioinformatics dataset

Oleksii Stroganov, BSc.

- Junior Biotechnologist at YURiA-PHARM
- Maintainer of multiple open-source projects on GitHub
- Interests: Metabolic engineering, $LaTeX$, pharmaceutical platform development

What is really a computer?

You use a computer through **windows, icons, and clicks** — the Graphical User Interface, GUI.

But it is not the computer.

It's one **friendly, opinionated wrapper** over the real thing.

Someone decided:

- which buttons you get to press
- which folders are visible
- which commands are hidden

The computer itself doesn't care about any of that.

To talk to it directly, we use the shell

A **shell** is a text-based interface that lets you *instruct* the computer directly. You type what you want, the computer does it.

- no button was drawn — you don't need one
- no menu decided what's possible — everything is
- no click was tracked — just your keystrokes

Today you'll leave the GUI behind and learn the commands of computer magic.

What can you control?

Today you'll learn two types of spells:

Transform reality	Travel through time
BASH — your main language	Git — when you need to work with time
Create, Edit, Delete, Control	*Clone, Snapshot, Diverge, Recover*
Any data you have	Any git repository

Both are text. Both live in the same shell.

Together they cover 90% of what a working scientist or engineer does in a day.

Why do you need to learn it?

It makes you stronger.

But seriously, by the end of today you can:

1. Know the foundations of Bash & Git.
2. Understand how to work in a common Linux/macOS environment.
3. **fetch, inspect, and process** a real biological dataset.
4. Make your project **version-controlled**.
5. **Write a short report** to track your progress.
6. Learn how to connect to external servers.

You will be someone who can sit down at any terminal and not panic.

Course prep

To start with the course, let's first prepare. You will need a browser tab pointed at the right URL:

For all participants	Hardcore enthusiasts
https://bashland.org/	https://bashland.org/hard
guided intensive	open challenge, separate task repo

Login

Both are password-protected. Grab your key:

- **login:** `student`
- **password:** `BashLand2026!`

*Before typing: switch keyboard layout to **English** and make sure **Caps Lock** is off.*

For a command reference during the day (no login needed):
https://bashland.org/docs

Today's subject: Synechocystis sp. PCC 6803

A tiny **cyanobacterium** — a photosynthetic bacterium.

- one of the first bacteria to be **fully sequenced** (1996)
- textbook model for how **oxygenic photosynthesis** works
- the ancestral relative of **plant chloroplasts**
- widely engineered for **biofuels** and **biopharmaceuticals**

Genome: ~3.6 Mbp, one circular chromosome + plasmids. Encodes ~3,600 proteins.

Today you'll analyze its **proteome** — the list of every protein it knows how to make.

What exactly are we analyzing? FASTA.

FASTA is a plain-text format for **DNA, RNA, or protein sequences**.

```
>WP_010871214.1 photosystem II q(b) protein [Synechocystis]
MTATLERRESQSLWKRFCEWITSTENRLYIGWFGVLMIPTLLTATSVFIIAFIAAPPV
LYGNNIISGAIIPTSAAIGLHFYPIWEAASLDEWLYNGGPYQMIVCHFLLGVACYMGR
...

>WP_010871215.1 biosynthetic arginine decarboxylase [Synechocystis]
MSTNLPSKPLLQLAKSLSAAVLAFPHPQGYLNQFEQDDLFLQMLRALQNQFHNVDIAA
...
```

- a header line starts with `>` — sequence ID + description
- following lines are the sequence itself
- **one `>` line = one protein**

That last fact is the trick to counting.

Part 1 — Transform reality

your first spells in bash

Where am I?

```
pwd                 # print working directory
```

You see something like:

```
/home/student
```

That's your **home directory**. Everything you make today lives here.

Refresh the browser tab and everything is *poof* — gone.

What is here?

```
ls                  # list files
ls -l               # long format: perms, size, modified
ls -la              # also hidden files (start with .)
```

The `-l` and `-a` are **flags**. Flags modify a command.

Every command has a helper spell:

```
ls --help
```

Read it. It (almost) never lies.

Move around

```
mkdir work          # summon a directory
cd work             # step into it
pwd                 # confirm

cd ..               # up one level
cd work             # back in
cd ~                # home
cd -                # previous location
```

`~` = shorthand for your home directory.

`..` = the directory above the current one.

Make files

```
touch notes.txt               # empty file
echo "hello"                  # prints "hello"
echo "hello" > organism.txt   # writes to a file
echo "world" >> organism.txt  # appends to a file
cat organism.txt              # show contents
```

- `>` **overwrites** a file
- `>>` **appends** to it

These little arrows are the workhorse of the whole shell.

Look at files

```
cat  README.md                # small: dump to screen
less README.md                # long: pager. q to quit.
head README.md                # first 10 lines
tail README.md                # last 10 lines
head -n 3 README.md           # first 3
wc   README.md                # count lines, words, bytes
wc -l README.md               # only lines
```

Try `cat README.md` now — the task for today is written there.

Part 2 — Travel through time

meet git

Why git?

- **undo** any mistake, all the way back
- see **what changed** and when
- **compare** two versions of a file
- try an idea on a **branch** without losing the working version

Real research and real code both use git.
Today you'll use it too, from the first file you create.

Initialize a repo

Inside your `work` directory:

```
git init              # turn this dir into a repo
git status            # what's tracked, what isn't
```

`git status` is the command you'll type most often.

Type it after **every** other git command until it feels boring.

Track a file

```
git add organism.txt         # stage
git status                   # now "staged"
git commit -m "first notes" # save a snapshot
git log                      # see history
git log --oneline            # compact
```

`add` puts a file into the **staging area**.

`commit` snapshots whatever is staged.

Together they mean: *"take a picture of this state, label it, keep it forever."*

See what changed

```
echo "phylum: Cyanobacteria" >> organism.txt
git status                    # "modified"
git diff                      # show me the change

git add organism.txt
git commit -m "add phylum"
git log --oneline
```

- `git status` = *what* changed
- `git diff` = *what the change actually is*

Break

20 minutes

Stretch, water, refill your mana.

Part 3 — Fetch the data

grep meets wget

The task is in the README

You already saw the task with `cat README.md`.
Somewhere in there is a **URL** — the location of the Synechocystis protein FASTA on NCBI.

Rule of the day:

> **never type a long URL by hand.**

If a computer put it in a file, ask the computer to read it back.

Meet grep — find matching lines

`grep` prints lines that match a pattern.

```
grep https README.md
```

Every line containing "https" — the URL lines.

Now extract *just* the URL, no surrounding text:

```
grep -o 'https://[^ ]*' README.md
```

- `-o` = **only** the matching part
- `[^ ]*` = any characters except a space (= the URL alone)

Feed grep's output into wget

Save it into a variable:

```
URL=$(grep -o 'https://[^ ]*\.faa\.gz' README.md)
echo $URL
```

`$(...)` runs a command and hands you its output.

Now use it:

```
wget "$URL"
```

Zero manual typing. The computer read it, remembered it, and handed it to `wget`.

Install what you need with sudo

You might notice `wget` isn't installed yet:

```
wget: command not found
```

Install it:

```
sudo apt-get update
sudo apt-get install -y wget
```

`sudo` runs a single command as **root** — the administrator.
Here it's locked to `apt-get` only. Try `sudo cat` — it refuses.

Unpack the data

```
ls -lh                      # ~700 KB, compressed .gz

file GCF_..._protein.faa.gz    # what is this?

gunzip -k GCF_..._protein.faa.gz   # -k keeps the .gz

mv GCF_..._protein.faa synechocystis.faa

ls -lh                      # ~1.4 MB, uncompressed
```

Rename because life is too short to type `GCF_000009725.1_ASM972v1_protein.faa` fifty times.

Tell git to ignore the big file

Before touching git again:

```
echo synechocystis.faa    >  .gitignore
echo synechocystis.faa.gz >> .gitignore

git add .gitignore
git commit -m "ignore raw FASTA"

git status                # clean, even with the big file present
```

`.gitignore` = one pattern per line. Files matching are invisible to git.

Part 4 — The analysis

more grep, plus pipes, cut, tr, seqkit

Count proteins with grep

Every protein starts with a `>`. So the total protein count is the number of `>` lines:

```
grep "^>" synechocystis.faa | head    # look at some
grep -c "^>" synechocystis.faa        # count them
```

- `^` in the pattern = *"start of the line"*
- `-c` = *count instead of print*

Your first real number. Write it down for the report.

Chain commands with pipes

`|` sends the output of one command as input to the next.

```
grep "^>" synechocystis.faa | grep -i ribosom | head
```

Read it left to right:

1. keep header lines
2. of those, keep the ones mentioning "ribosom" (case-insensitive)
3. show the first 10

This is the shape of most bash work: **short tools, chained together.**

Save your findings

Use `>` from Part 1:

```
grep "^>" synechocystis.faa | grep -i ribosom > ribosomal.txt
wc -l ribosomal.txt
```

Then commit as you go:

```
git add ribosomal.txt
git commit -m "ribosomal list"
```

One command per pipe stage. One commit per finding.

The psbA gotcha

psbA — the classic photosystem II reaction-center protein. Search for it:

```
grep "^>" synechocystis.faa | grep -i psba
```

Zero hits. RefSeq labels it *photosystem II q(b) protein*.

Broaden the search:

```
grep "^>" synechocystis.faa | grep -i "photosystem ii" > psba.txt
git add psba.txt
git commit -m "psba family attempt"
```

Real biology, not a bash problem.

cut — pick columns

```
echo "apple,banana,cherry" | cut -d, -f1        # apple
echo "apple,banana,cherry" | cut -d, -f2,3      # banana,cherry
```

- `-d` = delimiter
- `-f` = which field(s)

A FASTA header's ID is the **first space-separated word**:

```
head -n 1 synechocystis.faa | cut -d' ' -f1     # >WP_010871214.1
```

tr — replace or delete characters

```
echo "hello" | tr -d 'l'          # heo   (delete l's)
echo ">WP_123.1" | tr -d '>'      # WP_123.1
```

- `-d` = delete these characters

Small, sharp, single-purpose.

Compose: extract every protein ID

```
grep "^>" synechocystis.faa \
  | cut -d' ' -f1 \
  | tr -d '>' \
  > all_ids.txt

wc -l all_ids.txt

git add all_ids.txt
git commit -m "all protein ids"
```

1. keep header lines
2. keep only the first word (ID with `>`)
3. drop the `>`
4. save to file, commit

seqkit — the bioinformatics shortcut

`seqkit` speaks FASTA natively:

```
seqkit stats synechocystis.faa          # one-line summary
seqkit stats synechocystis.faa.gz       # works on the .gz too
```

Output:

```
file             format  type     num_seqs    sum_len  min_len  avg_len  max_len
synechocystis..  FASTA   Protein     3,576  1,117,688       26    312.6    4,199
```

Everything you just spent an hour computing, in one line.

Now you know **why** bio tools exist — and what they're really doing.

Write `report.md`

Use a heredoc to make a starter:

```
cat > report.md <<'EOF'
# Synechocystis sp. PCC 6803

- assembly:        GCF_000009725.1
- source:          NCBI
- total proteins:  <fill in>
- psbA family:     <see psba.txt>
- ribosomal count: <see ribosomal.txt>
EOF
```

Fill in the numbers, then:

```
git add report.md
git commit -m "first draft of report"
```

Part 5 — SSH

connect to a remote machine

What is SSH?

Secure **Sh**ell — the standard way to open a shell on a remote computer over the internet.

- encrypted end-to-end
- authenticated (password or key)
- works from any OS

Today's playground server:

- **host:** `35.184.182.150`
- **username:** anything you like (try your first name)
- **password:** `qwerty`

SSH on Windows

Windows 10 (1809+) and Windows 11 include OpenSSH by default.

Open **PowerShell** or **Command Prompt**:

```
ssh <name>@35.184.182.150
```

If it says "command not found":

Settings → **Apps** → **Optional features** → **Add a feature**
→ search **OpenSSH Client** → install.

Or grab **PuTTY** (putty.org) — GUI alternative.

SSH on macOS

macOS ships with OpenSSH. Open **Terminal.app** (`Applications` → `Utilities` → `Terminal`).

```
ssh <name>@35.184.182.150
```

That's it. No install needed.

Tip: iTerm2 (iterm2.com) is a nicer terminal. Optional.

SSH on Linux

You almost certainly already have it. Test:

```
which ssh
```

If missing (rare):

```
sudo apt-get install -y openssh-client   # Debian, Ubuntu
sudo dnf install -y openssh-clients      # Fedora
```

Connect:

```
ssh <name>@35.184.182.150
```

First connection

The first time you connect:

```
The authenticity of host '35.184.182.150' can't be established.
ED25519 key fingerprint is SHA256:xxxxx.
Are you sure you want to continue connecting? (yes/no)
```

Type `yes` and Enter. The server is remembered so you get warned if it ever changes.

Then enter the password: `qwerty`

To disconnect later: type `exit`.

Appendix — Wrap up

what "done" looks like

The story of the session

```
git log --oneline --graph --all
```

Should read like a small narrative:

```
* 9f2a1c  first draft of report
* 8c4b3d  all protein ids
* 7a1e5c  ribosomal list
* 6d3f8e  psba family attempt
* 5f2a1c  ignore raw FASTA
* 4c4b3d  add phylum
* 3a1e5c  first notes
```

One commit per logical step. That's the goal.

Bonus: try a branch

```
git switch -c try-photo
grep "^>" synechocystis.faa | grep -i photo > photo.txt
git add photo.txt
git commit -m "explore photosynthesis-related"

git switch main
ls                        # photo.txt is gone here

git merge try-photo       # bring it in
```

Branches let you try ideas without disturbing what works.

A parallel timeline. When you like the outcome, merge it back.

Want to keep practicing?

Keep the terminal alive on your own machine.

- **Windows** — install **WSL** (Windows Subsystem for Linux):

  ```
  wsl --install
  ```

  A real Ubuntu shell inside Windows.

- **macOS** — you already have Terminal.app. Same tools.
- **Linux** — you know.

Everything you learned today works there.

Thank you!

You may no longer be afraid of the terminal.